THEISZ GYÖRGY

BASIC TANÁCSADÓ C-16 PLUS/4

Theisz György

BASIC TANÁCSADÓ

C–16
PLUS/4

Műszaki Könyvkiadó, 1987

Lektorálta: **Könözsy László**

© Theisz György, Budapest, 1987

ETO: 800.92 BASIC

ISBN: 963 10 7321 1

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Szűcs Péter igazgató
Felelős szerkesztő: Votisky Zsuzsa
A szedés a Műszaki Könyvkiadóban készült
87/346 Sylvester J. Nyomda, Szombathely
Felelős vezető: Hanuszek Béla igazgató
Műszaki vezető: Kőrizs Károly
Műszaki szerkesztő: Orlai Ernő
A kötést tervezte: Kováts Tibor
A könyv ábráit rajzolta: Szabó Kálmánné
A könyv formátuma: B5
Ívterjedelme: 11,84 (A5)
Ábrák száma: 15
Papír minősége: 80 g ofszet
Betűcsalád és -méret: New Times, 10/11 p.
Azonossági szám: 61 510
MŰ: 4065—h—8790
Készült az MSZ 5601 és 5602 szerint

TARTALOMJEGYZÉK

Előszó 7
Bevezetés 9
1. A billentyűzet 11
1.1. A számítógépek billentyűzete 11
1.2. A számítógépesk billentyűkezelése 12
1.3. A C16 billentyűzete. Különleges billentyűk 13
2. A képernyő 16
2.1. A számítógépek képernyőkezelése 16
2.2. A C16 képernyőkezelése 17
2.3. Képernyőszerkesztő 19
3. A kulcsszavak típusai 22
3.1. Parancsok 22
3.2. Utasítások 23
3.3. Kulcsszócsoportok 23
3.4. Műveletek, függvények 23
4. A programozásról 25
4.1. A programozás fázisai 26
4.2. A célkitűzés, a feladat megfogalmazása 26
4.3. A megoldás algoritmusa 27
4.4. A program moduljai 29
4.5. A kódolás 36
4.6. Hibakeresés, belövés 38
5. Amit a BASIC-ről érdemes tudni 40
5.1. A változó. Változótípusok és tárolásuk. Kifejezések 40
5.2. Feltételvizsgálat 43
5.3. A FOR—NEXT-ciklus működése; a veremtár 44
5.4. A DO—LOOP-ciklus lehetőségei 45
5.5. Szubrutinok, szubrutinszerkezetek 46
5.6. A program adatai: DATA—READ—RESTORE 47
5.7. Adatbevitel: GET, INKEY$, INPUT 48
5.8. Numerikus függvények, felhasználói függvények 49
5.9. Szövegfüggvények 50
5.10. Speciális függvények 51
5.11. Hibakezelés a program futása közben 51
6. A program és az adatok tárolása 53
6.1. A program a tárban 53

6.2. A változók helye a tárban 55
6.3. A programjavítás következményei 56
6.4. A változók kezelése 56
7. A C16 kulcsszavai 57
8. A C16 monitorja 85
9. Néhány BASIC-trükk 88
9.1. Valós—egész számkonverzió 88
9.2. A bináris és a decimális aritmetika 89
9.3. Aritmetika logikai operátorokkal 89
9.4. IF nélkül is lehet 90
9.5. A FOR—NEXT-ciklus ezeket is tudja 91
9.6. Kerüljük a GOTO használatát! 92
9.7. Használjuk a DO—LOOP-ciklust! 93
9.8. Mi van a tárban? 94
9.9. Programok összetoldása (MERGE) 96
9.10. A program lassításának „mesterfogásai" 96
10. Feladatmegoldások 99
11. Mintaprogramok 106
Epilógus 120
Függelék 121
Irodalomjegyzék 128

ELŐSZÓ

Miután nagyobb mennyiségű COMMODORE 16 és PLUS/4 típusú gépet hoztak kereskedelmi forgalomba, sőt az általános iskolákba is sok ilyen gép került, a hazai géppark belső szerkezete jelentősen változott. E gépek iskolai elterjedése indokolja egy olyan könyv megjelenését, amely a programozás legalapvetőbb ismereteinek elsajátításához a COMMODORE 16 BASIC-jét használja.

Mivel a gép elég megbízhatónak és tartósnak ígérkezik, így érdemes a számítástechnikai ismeretek alapozását jelentős részben erre tervezni. Ez a bevezetés kétoldalú folyamat, így e könyv is kettős célt szolgál. Segíteni az önálló tanulást és – kissé később – a programírást, másrészt szakmai-módszertani segítséget nyújtani azon tanároknak, akik – bár maguk is kezdők – vállalják ezt a népszerűsége ellenére is nehéz feladatot. A könyv szerkezete egy módszertani lehetőség a sok közül. Természetesen minden leírás sugall valamilyen tárgyalási módot, még akkor is, ha nem ilyen célzattal készült. A feladatgyűjtemények és a példatárak esetében kissé bonyolultabb a helyzet. Ezek ugyanis csak a jó pedagógiai munka mellett biztosítják a „gazdaságos" tanulást: az összes feladat megoldása könnyen vezethet elaprózódásra, lényegtelen dolgok túlhangsúlyozására, bizonyos feladatok kihagyásának viszont felületesség, az alapvető ismeretek hézagossága lehet a következménye.

Sok könyv tartalmaz a tanulási folyamat kritikus pontjaira vonatkozó utalást. Ezek – mint a napi pedagógiai munka tapasztalatai – rendkívül érdekesek, de ne feledjük, hogy az ilyen pontok erősen függenek az alkalmazott módszerektől és természetesen a tanulóktól is.

Részben ezért, de mivel szinte minden ponton tapasztaltam nehézségeket, az ilyen utalásoktól tartózkodtam. Úgy vélem, hogy minden olyan témakörben adódhat nehézség, amelyet nem kezeltünk eléggé korrektül, akár a tanulásban, akár a tanításban.

A címek mellett általában egy-egy mottó segíti a megfelelő rész témakörének a behatárolását. Ezek Artur Block Murphy törvénykönyve, avagy „Miért romlik el minden?" című munkájából való idézetek. Tudományos alapjuk nincs, így tilos ezeket komolyan venni, de figyelmen kívül hagyni őket dőreség lenne.

A könyv talán elég mélyen tárgyalja a témát ahhoz, hogy a kezdő számára a középfokú könyvek megértéséhez is biztos támaszt nyújtson.

Ezen célok eléréséhez kíván fegyelmezett és pontos munkát, jó szórakozást:

A szerző

BEVEZETÉS

A Commodore cég számítógépei igen népszerűvé váltak az elmúlt években, így érthető, hogy sokan éppen a C16-os gép segítségével kezdenek ismerkedni a számítógépekkel. Elég a gép bővített BASIC interpreterét, a beépített monitorát és a Commodore 64 lemezkezelésével felülről kompatibilis lemezkezelését említeni ahhoz, hogy közkedveltsége érthető legyen. Mindezek egyben biztosítékot is jelentenek a számítástechnikával így ismerkedők számára a továbblépésre, azaz nem kell félni attól, hogy az Olvasót egy — esetleg túlságosan is rövid — zsákutcába vezetné e könyv.

A könyv nem kíván semmilyen számítástechnikai vagy matematikai előismeretet. Az a néhány rész, amely középiskolai matematikában szereplő ismereteket használ, minden következmény nélkül elhagyható. Ezek elsősorban a haladóbbak és a tisztesség kedvéért kerültek a könyvbe: a BASIC ugyanis ezeket is tudja.

Gépünk BASIC-nyelve kiválóan tud adatállományokat (file) is kezelni. Ezen sajátosságainak tárgyalása azonban igen messzire vezetne, ráadásul ezek még tágabb értelemben sem tartoznak a BASIC alapozásának a tárgykörébe. Aki ilyen szintre jutva szűknek találja a könyv kereteit, az a készülékek (számítógép, magnetofon, lemezegység, nyomtató) kézikönyveiből gyűjtheti össze a szükséges ismereteket, de bátran támaszkodhat a Commodore 64 hasonló témájú ismertetőire, leírásaira is, mivel a C16-os ebben a tekintetben is felülről kompatibilis elődjével.

A számítógépeket a továbbiakban három fő működési egységből állónak gondoljuk: a rendszer „lelke" a *mikroprocesszor,* működése során a *tár* adatain végez műveleteket, vagyis a *be-,* ill. *kimeneti egységek* segítségével adatokat olvas be vagy ír ki. A tár egy része rögzített tartalmú; itt tárolódnak azok az adatok, amelyek — többek között — a BASIC értelmező működését írják elő. A tár írható része tartalmazza a működés során szükséges változó adatokat, így pl. az általunk írt programot, ennek adatait, a gép pillanatnyi állapotát. A be-, ill. kimeneti egységek bizonyos határok között önállóan is működnek: programfutás közben pl. a mikroprocesszortól függetlenül történhet a képernyő képének előállítása vagy a billentyűzet figyelése, esetleg valamilyen külső eszköz (periféria) kiszolgálása. Természetesen ilyenkor is — a képgenerálást kivéve — a processzor kezdeményezi az adatforgalmat. Ha a gép egyetlen „intelligens" áramkört tartalmaz, akkor az adatmozgást ez végzi, azonban mindezekből BASIC szinten tulajdonképpen semmi sem látszik.

A számítógép felsorolt egységeit három vezetékrendszer, ún. busz köti össze. A vezérlőbusz segítségével jelöli ki a processzor azt az egységet, amivel kommunikálni akar, jelzi hogy milyen műveleteket kezdeményez, és még néhány egyéb (például időzítési) jelzés is ezen a buszon továbbítódik. A címbusz az egység kiválasztott

részét jelöli ki, így pl. a tár valamelyik byte-ját. Az adatbusz a gépen belüli adatforgalmat bonyolítja le; ezen továbbítódnak az egyes egységek között az adatok.

Nyilvánvaló, hogy a fentebb leírtak a számítógépeknek egy igen egyszerű modelljét adják, de ennél többet nem szükséges tudnunk ahhoz, hogy a továbbiakat megértsük.

1. A BILLENTYŰZET

„Nem lehet tudni, hogy milyen mély egy pocsolya, míg bele nem lépünk."

Miller törvénye

1.1. A SZÁMÍTÓGÉPEK BILLENTYŰZETE

„Könnyen érthető, jól működő tévedés hasznosabb, mint a bonyolult, érthetetlen igazság."

Sacc második posztulátuma

A személyi számítógépek egyik alapvető bemeneti egysége a billentyűzet. Bekapcsoláskor a gép a billentyűzetről várja üzenetünket, majd ennek megfelelően működik tovább.

Néhány botkormánnyal működő programtól eltekintve általában a billentyűzet segítségével visszük be a gép által kért adatokat programfuttatás közben is. A billentyűzet a legalapvetőbb eszköze a párbeszédes működésnek.

A számítógépek billentyűzetét úgy alakítják ki, hogy a gép számára szükséges alapvető jeleken kívül néhány egyéb — speciális funkciójú — jelet is bevihessünk a gépbe. Ha egy programlistát megnézünk, akkor abban a betűkön, számjegyeken, szóközön, írásjeleken és műveleti jeleken kívül — ezek az írógép billentyűzetén és a gépelt szövegekben is megtalálhatók — csaknem biztosan találunk néhány olyan karaktert (nyomtatott jelet), aminek jelenléte első látásra nem nyilvánvaló: funkciójuk megértése némi számítástechnikai — ill. konkrét gép- — ismeretet feltételez.

Egy számítógép billentyűzete ránézésre nagyon hasonlít az írógépekére, bár néhány különbséget könnyen találhatunk. Azon kívül, hogy a számítógépek legnagyobb része csak az angol ábécé betűit „ismeri", fontos tudni és megjegyezni, hogy a számítógép külön nulla és egyes számjegyet használ, amik soha sem helyettesíthetők az „O", ill. „l" betűkkel, mert ezek jelentése a gép számára egészen más. Így tehát számíthatunk rá, hogy a gép egy ilyen elírás hatására hibajelzéssel reagál, aminek külön örülhetünk, hiszen így észrevehetjük a hibát. De sajnos az is előfordulhat, hogy a hibásan begépelt sort a gép — természetesen egyáltalán nem úgy, ahogyan azt elvárnánk — értelmezni tudja, és ennek hatására, esetleg csak jóval később, egészen érthetetlenül kezd működni. Érdemes tehát a számítógéppel való ismerkedést a billentyűzet alapos tanulmányozásával kezdeni.

1.2. A SZÁMÍTÓGÉPEK BILLENTYŰKEZELÉSE

„Minden törvény csak utánzata a valóságnak."

Lilly metatörvénye

A számítógépek billentyűzete és a mikroprocesszor közötti adatátvitel fizikai lehetősége általában minden számítógépben véglegesen adott. A processzor persze csak rövid ideig — a működés időtartamának kicsiny hányadában — teszi lehetővé logikailag is az adatátvitelt. Mindez rendkívül rövid ideig tart és nagyon sokszor ismétlődik, így a felhasználó úgy érzi, hogy a billentyűzet folyamatosan a rendelkezésére áll. Ezt a benyomását csak erősíti az a tény, hogy adatbevitel közben a gép valóban csaknem kizárólag a billentyűzet kezelésével tölti idejét.

Ahhoz, hogy a billentyűzet működését megérthessük, feltétlenül kell néhány szót szólni a be-, ill. kimeneti egységekről.

Az eléggé nyilvánvaló, hogy a számítógép működtetésének alapvető célja az információszolgáltatás. Az általunk várt adatokat általában a mikroprocesszor állítja elő a tárban levő adatokból, és — célszerűen — ugyancsak a tárban helyezi el ezeket. Így tehát az adatkivitelt a tár és a külső egység (pl. a magnetofon) között érdemes szervezni. (Adatátvitel közben ugyanis már nem biztos, hogy marad elegendő idő az adatfeldolgozásra.)

Az adatbevitel teljesen hasonló módon, de fordított irányban történik. A gép számára szükséges — feldolgozandó — adatot mindenképpen célszerű tárolni, hiszen általában több adatot kell bevinni, majd ezeket lényegében egyszerre kell feldolgozni. Közben persze a korábban bevitt adatokat tárolni kell.

Ezek után nézzük röviden a billentyűzetkezelés logikáját!

1. A processzor vizsgálja a billentyűzetet, hogy melyik billentyű(k) van(nak) lenyomva.

2. Ha a lenyomott billentyőkombinációhoz tartozik belső kód, akkor ezt előállítja, különben visszatér és folytatja a billentyűkezelés idejére felfüggesztett tevékenységet.

3. Az előállított belső kódot a tár megfelelő helyére tölti és visszatér a felfüggesztett tevékenységre.

Mindezek többféle hardver és szoftver segítségével is megoldhatók, számunkra elsősorban azért érdekes ennek ismerete, mert így érthető, hogy a program futása miért szakítható meg egy (speciális) billentyű lenyomásával, ill. hogy miért emlékszik bizonyos esetekben olyan billentyűkre is, amiket pl. programfutás közben nyomtunk le. Erről később még lesz szó.

1.3. A C16 BILLENTYŰZETE. KÜLÖNLEGES BILLENTYŰK

„Semmi sem olyan csábos közelről, mint messziről. Másrészt semmi sem olyan csábos messziről, mint közelebbről."

A megfigyelésre vonatkozó törvény

Számítógépünk billentyűinek többsége karakterbillentyű, azaz olyan billentyű, amit alapállapotban leütve a neki megfelelő karakter megjelenik a képernyőn. Természetesnek találjuk azt is, hogy a megjelenő karakter nagybetű, szám, olyan jel lesz, mint ami a billentyűn látható, és ha a billentyűn két jel is szerepel, akkor — ez szintén természetes — az alul levő kerül a képernyőre. Csekély írógépes gyakorlattal is rájövünk, hogy a felső jel (a számoknál és néhány írásjelnél) az írógép váltójának megfelelő SHIFT-billentyűvel érhető el: ezt lenyomva tartjuk, mialatt a másik billentyűt (is) lenyomjuk. Próbálkozásunk természetesen sikeres lesz, de örömünkben ne feledkezzünk el arról, hogy egy fontos következtetést levonjunk: a SHIFT-billentyű önmagában való lenyomása hatástalan, azaz ez a billentyű csak módosítja a vele egyidejűleg lenyomott másik billentyű hatását.

Ezek után önkéntelenül adódik a következő kérdés: Mi történik, ha a SHIFT-tel együtt egy betűt nyomunk le? Nyugodtan kipróbálhatjuk, biztosan nem okozunk kárt gépünkben. (Itt érdemes megjegyezni, hogy billentyűzetről, vagy bármilyen más módon bevitt szoftverrel gépünket *nem lehet* tönkretenni: újbóli bekapcsoláskor a gép normálisan indul — előéletéről semmit sem tud. Mindez azt is jelenti, hogy ha a gépünk meghibásodik, akkor azt nem a szoftver és nem is az a barátunk, családtagunk stb. okozta, aki éppen a billentyűzet előtt ül.) Az eredmény egy furcsa (grafikus) jel lesz, ami megintcsak nem meglepő, hiszen ugyanezt a jelet láthatjuk a leütött billentyű előlapjának a jobb oldalán. Ez viszont újabb kérdést vet fel: Hogyan érhetjük el a bal oldali jelet? Erre a SHIFT-billentyűhöz nagyon hasonló ún. COMMODORE-billentyű (jele: C=) használható. Önmagában ez a billentyű is hatástalan, csak a vele egyidejűleg lenyomott billentyű hatását módosítja. Ha valaki ezen elnevezéssel kapcsolatban gyanakodni kezd, akkor csaknem biztosan igaza lesz; ilyen másodlagos SHIFT funkció általában nem létezik más típusú gépeken.

Ha már érteni véljük ezen két billentyű hatását, akkor ássunk egy kicsit e kettősség mélyére! A C= és a SHIFT-billentyű egyidejű lenyomása hatására a képernyő furcsa módon megváltozik. A nagybetűk kisbetűvé válnak, a SHIFT-tel bevitt grafikus karakterek helyén a billentyűnek megfelelő nagybetű jelenik meg, de változatlanok maradnak a számok, a műveleti és írásjelek. (Sajnos igen nehéz zártan megadni azt, hogy ilyenkor milyen karaktercsere megy végbe az egyes esetekben, de erről az Olvasó maga is meggyőződhet, ha pl. az „=" jelet billentyűzi be a SHIFT- és a C= billentyűvel együtt is, majd lenyomja a C= és a SHIFT-billentyűt együtt.) Az említett két billentyű ismételt lenyomásának hatására a képernyő eredeti állapota visszaáll. A képernyő kétféle állapota a megjeleníthető két karakterkészletnek megfelelően jön létre. Ezek egyike a nagybetűk ill. grafikus jelek, míg a másik a kisbetűk ill nagybetűk karakterkészlet. Bekapcsoláskor az előbbivel éled a gép. A kisbetűk ill. nagybetűk üzemmódot szokás írógépüzemmódnak is nevezni.

A SHIFT-funkció rögzíthető a SHIFT LOCK-billentyű lenyomásával. Ez a billentyű ismételt lenyomásáig marad érvényben, hatására a SHIFT-billentyűk funk-

ciója megfordul. Itt kell megjegyezni, hogy a gépen található két SHIFT-billentyű egyenértékű.

A harmadik ilyen módosító hatású billentyű a CTRL jelű (control) billentyű. Ennek hatóköre az előző kettőénél lényegesen kisebb: bizonyos vezérléseket végezhetünk a segítségével.

A karakterek színét a CTRL-billentyű és az 1-től 8-ig valamelyik számbillentyű, vagy a C= billentyű és egy, az előzőkben említett számbillentyű lenyomásával változtathatjuk. A CTRL-billentyű mellett az előlapra írt felső, a C= billentyű mellett az alsó színűvé válik a kurzor, és az összes ezután kiírt karakter színe is ilyen lesz.

A számítógép lehetővé teszi az ún. „inverz" karakterkiírást. Ez azt jelenti, hogy a kiírt karakter háttérszínű lesz, a karakterhely hátterének színe pedig az írás színére változik. Ezt a CTRL-billentyű, majd a 9-es billentyű lenyomásával kapcsolhatjuk be (RVS ON). A funkció kikapcsolódik a CTRL- és a 0 billentyű (RVS OFF), vagy a RETURN-billentyű, vagy az ESC és a 0 billentyű egymás utáni lenyomása hatására (l. még ESCAPE-funkciók!), ill. ha nyomtatás során az utasítás a sor nyomtatását befejezi.

A fentiekhez hasonló a karakterek villogtatását végző funkció. Bekapcsolni a CTRL- és a vessző-billentyű egyidejű lenyomásával lehet, kikapcsolható a CTRL- és a pont-billentyű egyidejű lenyomásával, de kikapcsolódik minden olyan körülmény hatására is, amit az inverz funkciónál felsoroltunk.

A CTRL- és az S-billentyű együttes lenyomása hatására a következő nyomtatási funkció előtt felfüggesztődik a gép működése, akár program-, akár parancsüzemmódban van. Így pl. felfüggeszthető a program listázása is. Ezt az állapotot a STOP-billentyűn kívül csaknem bármelyik billentyű lenyomásával megszüntethetjük, aminek hatására a folyamatban levő tevékenység folytatódik.

A C16 kezelési útmutatóiban (különböző nyelveken) általában csak a felsorolt CTRL-funkciókról tesznek említést, ráadásul gyakran egymásnak ellentmondóan. Éppen ezért érdemes csínján bánni a CTRL-billentyűvel, ugyanis igen meglepő dolgok történnek bizonyos betűknek ezen billentyűvel való együttes alkalmazásakor.

Az ESC- (escape: megmenekülni-) billentyű hatására a gép kikerül a képernyőszerkesztő felügyelete alól. (A képernyőszerkesztőről később lesz szó.) Ez azt jelenti, hogy a következőként leütött billentyű a megszokottól egészen eltérő következményeket vált ki, ezek azonban kivétel nélkül a képernyőre vonatkozó funkciók, így célszerűbb ezeket a képernyővel foglalkozó fejezet végén tárgyalni.

Hasonló okok miatt a kurzormozgató, az INST/DEL- és a CLEAR/HOME-billentyűkkel a képernyőszerkesztő tárgyalásakor fogunk foglalkozni. Itt csak annyit jegyezzünk meg, hogy ha a CLEAR/HOME-billentyűt lenyomjuk, akkor a kurzor a bal felső karakterhelyre kerül, de a képernyőtartalom nem változik meg. Ha ugyanezt a SHIFT-billentyű lenyomása mellett tesszük, akkor a képernyő törlődik is.

A RUN/STOP-billentyű lenyomása hatására megszakad a BASIC program futása, vagy az aktuális parancs végrehajtása, és a képernyőszerkesztő felügyelete alá kerül a gép. Mindez nem következik be, ha a gép éppen INPUT utasítás végrehajtásakor adatbevitelre vár. Ebben az esetben a SHIFT- vagy a C= billentyűvel együtt próbálkozhatunk, amikor is a bevitt szöveg egy lemezegységre vonatkozó parancs lesz, és — ha néhány egyéb feltétel is szerencsésen összejön — akkor valószínűleg megáll a programfutás. (Erre vonatkozóan később talál utalást az Olvasó.)

Szerkesztő üzemmódban a SHIFT- és a RUN/STOP-billentyűket egyidejűleg lenyomva megkezdődik a lemezen levő első program betöltése, ami a betöltődés után

azonnal elindul. (Ilyenkor tulajdonképpen egy DLOAD"★", majd egy RUN-parancs hajtódik végre.)

A különleges billentyűk közül a RETURN-billentyű okozza a legtöbb bosszúságot a kezdőknek. Funkciója a sorzárás, azaz ezzel a billentyűvel „hozzuk a gép tudomására", hogy az üzenetünk beírását befejeztük, kezdheti annak feldolgozását. A kezdők nagyon gyakran, de olykor még a haladók is elkövetik azt a hibát, hogy elfelejtik lezárni a sort, és a gépelés befejezése után várják, hogy a gép kitalálja gondolatukat. Erre a hibára fokozottan ügyeljünk, ugyanis később — programjavításkor és -szerkesztéskor — rengeteg bosszúságot fog okozni.

A COMMODORE hagyományoknak megfelelően a C16-ot is ellátták ún. funkcióbillentyűkkel. Ezek bekapcsolás után azonnal használhatók az alapértelmezésnek megfelelően, de lehetőség van átértelmezésükre is. Ez utóbbiról a KEY kulcsszónál, míg a billentyűk hatásáról a megfelelő kulcsszónál találunk részletes útmutatást.

f1: GRAPHIC	f4: SCNCLR <RETURN>
f2: DLOAD"	f5: DSAVE"
f3: DIRECTORY <RETURN>	f6: RUN <RETURN>
HELP: HELP <RETURN>	f7: LIST <RETURN>

A HELP az f8 funkciónak felel meg. A kulcsszó utáni <RETURN> sorzárást jelent, a többi esetben a sor még folytatható, ill. folytatandó.

Itt illik szólni a gép jobb oldalán található RESET-gombról, ami újraindítást végez. Hatására a gép a bekapcsolás utáni állapothoz hasonló állapotba jut, így — ha a tárat nem rontottuk el szerencsétlenül, akkor — a gép lényegében minden indítási tevékenységet újra végrehajt. Ezután minden olyan adat, ami a tár tartalmára vonatkozik, a bekapcsolás utáni tartalommal töltődik fel, úgy, mintha a tár üres lenne. Ez azonban nem fizikai, csak logikai törlést jelent.

2. A KÉPERNYŐ

„Két baj is csak a kezdet."

Murphy törvényének Kohn-féle folyománya

2.1. A SZÁMÍTÓGÉPEK KÉPERNYŐKEZELÉSE

„Ha nem tudod meggyőzni, zavard össze őket!"

Truman törvénye

A személyi számítógépek legfontosabb kimeneti egysége általában a képernyő. Ez a legtöbb esetben egy műsorvevő (fekete—fehér vagy színes) televíziókészülék, de igényesebb összeállításban monitor is lehet. Sajnálatos, hogy a kiváló képminőséget szolgáltató monitorok ára sokszorosa a házi célokra szánt számítógépek árának. Természetesen mindegy, hogy milyen készüléket használunk számítógépünkhöz: a kép szerkezete ettől független; a különbség a minőségben (felbontás, kontraszt, színek tisztasága stb.) jelentkezik.

A számítógépek képernyőkezelését nagyon változatos módszerekkel oldják meg. Ezek egyszerű áttekintése is nagyon messze vezetne, így csak a legfontosabb elvek tisztázására kell szorítkoznunk, ami egyúttal azt is jelenti, hogy más gépeknél az itt leírtaktól jelentősen eltérő képernyőkezeléssel is találkozhatunk.

A képernyőkezelés egyik szokásos módja, hogy a tárban lefoglalunk egy területet, ahol a képernyőn megjelenítendő karakterek kódjait tároljuk. Ezen tárterület tartalma alapján állítja elő a számítógép egy egysége a videojelet. A kódok karakteres formáját egy további tárterületen őrizzük.

A karakterkódok tárolására szolgáló terület nagysága általában 1 kbyte, ugyanis a képernyőkön legtöbbször mintegy 1000 karaktert jelenítenek meg. Ez a terület többnyire a számítógép írható, ill. olvasható tárában van: egyes gyártók ezt beleszámítják a gép RAM kapacitásába, mások nem. Az is előfordulhat, hogy ez a tárterület csak körülményesen érhető el a programozó számára, mert valamilyen módon ez a program által használt tár „mögött" helyezkedik el. A színmemória — ha van — ugyanilyen szerkezetű.

A karakterkódok alapján a videojelet egy karaktergenerátor állítja elő a legtöbb gépben, azonban ezt a (speciális) feladatot is olykor a mikroprocesszorra bízza egy-egy gyártó. Az ilyen gépek természetesen lassabban működnek, hiszen a processzor idejének egy (nem is kis) részében a videojel előállításával van elfoglalva. A komolyabb gépek tehát általában önálló karaktergenerátorral készülnek.

A karakterek képét általában egy másodlagosan kezelt fix tárban (karakter-ROM) tárolják. Ez a tárterület azonban esetenként máshol is kijelölhető, így a karakterkészlet cserélhető.

A képernyőkezelés egy másik szokásos módja, hogy a képernyőtartalom minden pontját külön-külön tároljuk. Ez azt jelenti, hogy minden pont állapotát (világít vagy sötét) egy-egy biten őrizzük. Figyelembe véve, hogy egy karakter egy 8×8-as pontháló, és byte-onként 8 bitet tárolunk. Egy képernyőtartalom mintegy 8 Kbyte területen fér el.

Könnyű átlátni, hogy ez a módszer lényegesen egyszerűbb karaktergenerátort igényel, viszont a kiírási műveletek lassabbak és nagyobb tárra van szükség.

2.2. A C16 KÉPERNYŐKEZELÉSE

> *„A kellően fejlett technika már megkülönböztethetetlen a mágiától."*
>
> *Clarke harmadik törvénye*

Számítógépünk öt különböző képernyőkezelést használ. Ezek közül egy karakteres, kettő grafikus és kettő osztott (karakteres és grafikus) képernyőt eredményez. Szerencsére ezek az üzemmódok elég szorosan kapcsolódnak egymáshoz, így igen egyszerűen átláthatók, használatuk némi gyakorlással elsajátítható.

Mielőtt az egyes üzemmódokat tárgyalnánk, érdemes néhány általánosabb fogalomra kitérni. Bekapcsolás után a képernyőnek három jól elkülöníthető része látszik:

1. A képernyő kerete; kódja: 4.
2. A képernyő háttere amire írunk, vagy grafikus üzemmódban rajzolunk; kódja: Ø.
3. Előtér, ami az írást, ill. a rajzot jelenti; kódja: 1.

A többszínű grafikánál a háttér és az előtér színén kívül két további szín használható.

4. Többszín—1, a rajzoláshoz használható második szín; kódja: 2.
5. Többszín—2, a rajzoláshoz használható harmadik szín, ami az előzőkhöz képest különleges tulajdonsággal bír átkapcsolás esetén; kódja: 3.

Ezeket a képernyőrészeket a C16 irodalma zónáknak is, forrásnak is nevezi.

A KARAKTERES KÉPERNYŐ

A C16 karakteres képernyője 25 sorból, soronként 4Ø, azaz összesen 1ØØØ karakterhelyből áll. Ehhez 1 Kbyte képernyő-RAM és ugyanekkora szín-RAM tartozik. (Itt jegyezzük meg, hogy a gép 16 Kbyte RAM-jából még további 2 Kbyte-ot használ a rendszer, így szabadon mintegy 12 Kbyte marad a felhasználó számára.) A karakter-ROM 2 Kbyte nagyságú: itt tárolódik a kétféle karakterkészlet.

A háttér és a keret színét egy-egy byte tárolja, így ezek mindenütt azonos színűek, a karakterek azonban mind különböző színűek lehetnek.

Itt ismét utalni kell e gép beépített képernyő-manipulációs lehetőségeire, amelyek bő választékban állnak a programozó rendelkezésére a bekapcsolás pillanatától kezdve. Ezek többnyire a karakteres képernyőn használhatók: leírásuk a képernyőszerkesztőhöz kapcsolódóan később következik.

A karakteres képernyő nullás grafikus üzemmódban (GRAPHIC Ø) aktív.

GRAFIKUS KÉPERNYŐK

Számítógépünk kétféle grafikus üzemmódban használható: nagyfelbontású grafika, ill. többszínű grafika.

A *nagyfelbontású grafika* könnyen átlátható: minden képernyőpontról egy-egy biten őrizzük, hogy az háttérszínű vagy előtérszínű. A képernyő ponthálózata — kézenfekvően — megegyezik a karakteres képernyőével. Így a grafikus képernyő vízszintesen, a 40 karakternek megfelelően: 320; függőlegesen, a 25 sornak megfelelően: 200 pontból áll. Ennek tartalmát egy 8 Kbyte nagyságú tárterületen kezeli a rendszer. A szín és a fényerő értékeit karakterhelyenként további 1—1 Kbyte-on tárolja *.

A fentiek szerint tehát a nagyfelbontású grafikára való áttérés után a szabad memóriaterület 10 Kbyte-tal csökken. Ezt a programírásnál feltétlenül tartsuk szem előtt.

A nagyfelbontású grafika pontjai a képernyő bal felső sarkából indulva 0-tól 319-ig, ill. 0-tól 199-ig számozódnak. Ez az 1-es grafikus üzemmód.

A *többszínű grafika* az előzőből úgy származtatható, hogy vízszintes irányban két-két szomszédos pontot egyetlen pontként kezelünk. Ennek színállapotát (háttérszín, előtérszín, többszín—1, ill. többszín—2) két, szintén szomszédos biten tároljuk. Nyilvánvaló, hogy a felbontás — igaz, hogy csak vízszintes irányban — felére csökken a nagyfelbontású grafikához képest.

Sajnálatos, hogy ezt a felbontáscsökkenést a vízszintes koordinátatartomány csökkentésével oldották meg, ennek ugyanis az a következménye, hogy a két grafikus képernyőn ugyanaz a grafikus utasítás különböző alakzatot eredményez. Ez különösen az ellipszis rajzolása esetén teszi nagyon áttekinthetetlenné a helyzetet. (A kétféle grafika közötti kapcsolatteremtésre sokkal elegánsabb megoldások is léteznek!)

A tár kezelése egyébként megegyezik a nagyfelbontású grafikánál leírtakkal. Ez a 2-es grafikus üzemmód.

Az elmondottakból következik az is, hogy a grafikus üzemmódra való áttérés a karakteres képernyő tartalmát érintetlenül hagyja.

OSZTOTT KÉPERNYŐK

Lehetőség van a C16-os gép alkalmazásánál a képernyő vegyes használatára is. Ilyenkor a megfelelő grafikus képernyő alsó 40 pontsora helyén a karakteres képernyő alsó 5 sora látszik.

Tudni kell, hogy a gép ilyenkor mind a grafikus, mind pedig a karakteres képernyőt teljes egészében kezeli, mindössze ezek megjelenítése osztott. Ennek megfelelően a tárigény megegyezik a teljes grafikus üzemmódokéval.

* Bár a grafikus képernyő tárolása a BASIC szintjén nem szokott gondot okozni, mégis érdemes tudni, hogy ez miként történik, ugyanis a BASIC néhány furcsaságát éppen ez okozza.

A grafikus képernyő a 6144—7167 (hexadecimálisan $1800—$1BFF) címtartományon a fényerőt, a 7168—8191 ($1C00—$1FFF) címtartományon a színeket tárolja, míg a bittérkép a 8192—16 383 ($2000—$BFFF) területen helyezkedik el.

16 Kbyte RAM esetén a grafika a legfelső 10 Kbyte; a szabad tárterület ez alatt — az eredeti helyén — marad.

A 64 Kbyte RAM-ot használó rendszer a BASIC szabad tárterületét mozgatja. Grafikus üzemmód bekapcsolásakor a BASIC-program és a változók a grafikus terület „fölé" kerülnek, az üzemmód törlésekor pedig visszakerülnek az alapállapotnak megfelelő helyre. Ebből adódik, hogy ilyenkor a grafikus tár alatti 2 Kbyte BASIC-ből nem „látszik", tehát a rendszer csak 48 Kbyte nagyságú területet használhat. (Az „eltakart" terület egyébként gépi kódú programok vagy akár képernyőtartalmak tárolására is használható.)

Ha osztott képernyő használata mellett a készült rajz egy része a szöveg helyére kerül, akkor ezt is kirajzolja a gép, csak a kijelzése marad el. A hiányzó rész is láthatóvá válik, ha a képernyőt a rajznak megfelelő (felbontású) egész képernyős grafikus üzemmódba kapcsoljuk.

Értelemszerű, hogy a kétféle grafikus üzemmódhoz kétféle osztott képernyős üzemmód tartozik. A nagyfelbontású a 3-as, a többszintű a 4-es grafikus üzemmód.

Figyelem! Ha osztott képernyőre való áttéréskor a képernyőt töröltetjük, akkor mind a grafikus, mind a karakteres képernyőtartalom elvész.

2.3. KÉPERNYŐSZERKESZTŐ

> *„Csak úgy lehet kitapasztalni a lehetőségek határait, ha átlendülünk rajtuk, a lehetetlenbe."*
> *Clarke második törvénye*

Minthogy egy begépelt program csak egészen kivételes esetekben hibátlan, és a programfejlesztés is elképzelhetetlen javítás nélkül, így a számítógépekbe beépítenek valamilyen szerkesztési lehetőséget. Erre alapvetően kétféle módszer használatos.

A sorszerkesztő aktivizálásakor valamilyen módon kijelöljük a javítani (szerkeszteni) kívánt sort, majd a szerkesztő szolgáltatásait használva végrehajtjuk a kívánt módosításokat. Ezután — jelezve, hogy a módosítást befejeztük — kilépünk a szerkesztő üzemmódból.

A képernyőszerkesztő a képernyőn (esetleg annak csak egy meghatározott részén) levő szöveg módosítását teszi lehetővé. A szerkesztés előtt gondoskodni kell arról, hogy a szerkesztendő szöveg a képernyő megfelelő helyére kerüljön, majd ezután a kurzor által kijelölt karakterpozíción elvégezhetjük a módosításokat. Ehhez természetesen szükség van a kurzor mozgatására, és néhány egyéb szerkesztési műveletre. A szerkesztés befejezése általában a sor lezárásával történik.

A C16-os számítógépen a teljes képernyőt használhatjuk szerkesztésre, azaz gépünk teljes képernyőszerkesztővel rendelkezik. Ennek működését csak néhány alapvető fogalom tisztázása után érthetjük meg, ugyanis a szerkesztésben a gép egyéb működési egységei is szerepet kapnak.

Arról, hogy a képernyőn 4Ø karakteres sorok jelennek meg, már volt szó. Bizonyára az is feltűnt az Olvasónak, hogy a gépelés a sor vége után is folytatható, ilyenkor a következő sorban folytatódik az írás. Ha mindez a legalsó sorban történt, akkor a képernyőtartalom egy sorral feljebb lép, és így jön létre a következő (üres) sor. A képernyő sorait fizikai soroknak nevezik, ugyanis ez a tagolás csak a megjelenítéskor érvényes.

A gép egy sorzáráskor a legutoljára begépelt vagy szerkesztett logikai sort fogja feldolgozni. Ez független a képernyőn látható fizikai sortól, hosszát a számítógépnek egy másik működési egysége korlátozza. Beíráskor ugyanis a feldolgozandó szöveg egy átmeneti tárolóba kerül, és csak ezután kezdheti a gép a feldolgozást. (Ennek oka a parancsüzemmódban végrehajtott képernyőtörlés. A kapcsolat a parancs tárgyalása után érthetővé válik.) Ennek az átmeneti tárolónak a hossza 88 byte, így a sorok hossza is legfeljebb ekkora lehet. (Ez vonatkozik a perifériák input műveleteire is, ami az adatállományok kezelésekor okozhat nehezen felismerhető hibákat.)

Ezután már könnyű leírni a képernyőszerkesztő vázlatos működését: a begépelt, ill. szerkesztett logikai sor sorzárás után az átmeneti tárolóba töltődik, ha belefér. Ha nem, akkor hibajelzést kapunk, és a gép a következő üzenetünket várja. Sikeres áttöltés esetén megkezdődik a bevitt szöveg feldolgozása, amivel az üzenetünk, ill. javításunk érvénybe lép.

Végül a szerkesztéshez használható szolgáltatásokról kell szólni. Ne keseredjen el az Olvasó, ha első olvasásra nem ért meg mindent, ennek ugyanis — részben — az az oka, hogy a szerkesztőt csak olyan bonyolultabb problémák megoldásánál kell igazán komolyan használni, melyekről csak jóval később lesz szó. Ezért ajánlom a programozásról szóló fejezet áttanulmányozása után a képernyőszerkesztő működésének az átismétlését.

Sor begépelésekor vagy szerkesztésekor egyaránt használható a négy kurzormozgató billentyű és a HOME-billentyű is. Nyilvánvaló, hogy a képernyőtörlés mindenféle szerkesztés befejeztét jelenti, hiszen a képernyőn nem marad értelmezhető szöveg. Ilyenkor csak sor begépelése kezdődhet.

Ha a SHIFT-billentyűt lenyomva tartjuk, és így zárjuk le a gépelés vagy szerkesztés alatt álló sort, akkor a sor feldolgozása elmarad, és a kurzor a következő logikai sor elejére áll.

Mód van a karakter törlésére és karakter(hely) beszúrására is. Erre az INST/DEL-(INSERT/DELETE-) billentyű szolgál; a beszúráskor a SHIFT-billentyűt is lenyomva kell tartani. Törléskor a kurzort közvetlenül megelőző karakter törlődik, beszúráskor a kurzortól jobbra keletkezik egy új karakterhely, ami átírás hiányában szóközként szerkesztődik a sorba. Természetesen a logikai sor kurzortól jobbra eső része a műveletnek megfelelően léptetődik.

Szólni kell az idézőjel-üzemmódról is, ami a képernyőszerkesztő használatát a kezdő számára kissé áttekinthetetlenné teszi, ugyanis „nyitó" idézőjel után a kurzorvezérlések nem hajtódnak végre, hanem a begépelt szövegbe szerkesztődnek. Karaktert törölni (DEL) azonban ilyenkor is lehet, de ennek sokszor olyan áttekinthetetlen helyzet az eredménye, hogy a gép viselkedését már nem tudjuk követni. Ilyenkor érdemes lezárni az utasítássort, majd a szerkesztést ismét elkezdeni, amikor is a kurzorral „átlépett" idézőjelről nem vesz tudomást a szerkesztő. Így kurzorvezérlést csak úgy tudunk beírni, hogy újra beütjük az idézőjelet, vagy ha a megfelelő karaktert töröljük, a helyére beszúrunk, és ezt a karakterhelyet írjuk felül a kurzorvezérlő karakterrel.

A fentiekből látszik, hogy a képernyőszerkesztő már alapállapotban is igen sokat tud, gazdaságos használatához komoly gyakorlat szükséges. A gyártó ennek ellenére hasznosnak tartotta az ESCAPE funkció beépítését, ami tovább bővíti a szolgáltatások körét.

ESCAPE-FUNKCIÓK

A Automatikus karakterbeszúrás a kurzor helyén. A sornak a kurzortól jobbra eső része a kurzorral együtt eggyel jobbra lép. A DEL-billentyű természetesen változatlanul használható, az INST (SHIFT-tel!) használata azonban értelmetlen.

B A képernyő aktív részének, a téglalap alakú képernyőablaknak a jobb alsó sarkát jelöli ki a kurzor aktuális helyén.

C Megszünteti a karakterbeszúró üzemmódot.

D Törli a kurzor logikai sorát és a kurzort a sor elejére állítja. A képernyő alsó része a törlésnek megfelelő számú fizikai sorral feljebb lép.

I A kurzor helyén egy üres fizikai sort hoz létre, és a kurzort ennek elejére állítja, miközben a kurzor sora és a képernyő alsó része egy sorral lejjebb lép. Mindezt a logikai soron belül is végrehajtja, ilyenkor a logikai sor egy fizikai sorral hosszabb lesz.

J A kurzort a logikai sor elejére viszi.

K A kurzort a logikai sor utolsó karakterére viszi.

L Megszünteti a képernyőgörgetés letiltását.

M Letiltja a képernyő görgetését. Ha ilyenkor a képernyő betelik, a listázás a képernyő elején folytatódik, míg a nyomtatás az utolsó karakterhelyen.

N Megszünteti a képernyőablakot, visszaáll az eredeti képernyőméret és törlődik a képernyő. (Ez utóbbi szükségessége nem nyilvánvaló, de a logikai sor fogalmát átgondolva könnyen megérthetjük.)

O Törli az idézőjel, a nem automatikus beszúrás (INSERT), az inverz- és a villogó üzemmódokat.

P Törli a logikai sor elejét a kurzor által kijelölt pozícióval bezárólag. A kurzor helyben marad.

Q Törli a logikai sort a kurzor által kijelölt karakterrel kezdve a sor végéig. A kurzor helyben marad.

R Törli a képernyőt és egy egykarakteres keretet állít be, így a létrejött képernyőablak 38×23-as lesz.

T A képernyőablak bal felső sarkát jelöli ki a kurzor aktuális helyén.

V A képernyőtartalmat egy (fizikai) sorral feljebb lépteti, miközben a kurzor helyben marad.

W A képernyőtartalmat egy (fizikai) sorral lejjebb lépteti, miközben a kurzor helyben marad.

X Hatástalan: törli az előzőleg kezdeményezett ESC hatását.

3. A KULCSSZAVAK TÍPUSAI

„Ahol nincs szükség övezetbesorolásra, ott tökéletesen beválik. Ahol nagy szükség volna rá, ott mindig csődöt mond."

McClaughry építésiövezet-törvénye

A BASIC nyelv néhány megegyezésben rögzített szó használatát teszi lehetővé a programozó számára. Minthogy ezek egy része gépfüggő, így kitalálták a BASIC nyelvjárás fogalmát, ami igen elegáns, de a helyzetet nem javítja: minden gép igazán hatékony alkalmazásához a gép saját nyelvét kell jól ismerni. Ez azt jelenti, hogy bizonyos általános ismereteken túl meg kell ismerkednünk a használt géptípus sajátságaival.

A tisztesség azt is megkívánja, hogy ezen a helyen szóljunk a BASIC nyelv egyik legsúlyosabb hibájáról. Arról, hogy ez a programozási nyelv a számítástechnikával nem foglalkozó, egyéb területeken dolgozó szakemberek számára készült azért, hogy adatfeldolgozási problémáikat saját maguk gyorsan és közvetlenül vihessék gépre. A nyelv tehát nem segíti a számítástechnikai szemléletmód kialakítását, ellenkezőleg, a program összetákolását, a próbálgatással való javítgatását támogatja. Ezért különösen fontos, hogy a *számítástechnika* iránt érdeklődő Olvasó megfelelő súlyt helyezzen a kulcsszavakról és a programozásról szóló részre.

3.1. PARANCSOK

„Semmi sem olyan egyszerű, mint amilyennek látszik."

A Murphy-törvény 1. folyománya

Bekapcsoláskor a legtöbb számítógép — így a C16 is — parancsüzemmódban éled. (Néhány rendszer ilyenkor a szervezésre vonatkozó információt is kér, azonban erről a kezelési útmutatók többnyire kielégítő mélységben írnak.)

A parancs olyan üzenet a gép számára, amit azonnal értelmeznie kell, majd el kell kezdenie a végrehajtást is.

Parancs végrehajtását általában parancsüzemmódban kezdeményezzük, és a legtöbb parancs végrehajtása után a gép — tetszőleges kezdőfeltételek mellett is — ismét parancsüzemmódba kerül. Sajnos az nem igaz, hogy minden parancs parancsüzemmóddal végződik, ugyanis ez néhányuk használhatóságát jelentősen korlátozná. Az ilyen parancsokat szokták — találóan ugyan, de kiveszőfélben levő kifejezéssel — közvetlen végrehajtású utasításnak nevezni.

3.2. UTASÍTÁSOK

„Ha már minden kísérlet csődöt mondott, olvasd el a használati utasítást!"

Cahn axiómája

Program írásakor az üzeneteink a program utasításai. Ezeket általában utasássoronként gépelhetjük be a BASIC parancsüzemmódjában, csaknem ugyanúgy, mint a parancsokat. A BASIC értelmező (interpreter) arról ismeri fel az utasítássort, hogy számjeggyel kezdődik. Az ilyen sorokat begépelés után nem kezdi értelmezni, hanem megfelelő alakra hozza ahhoz, hogy a programba beszerkessze, és ha ehhez elegendő hely van a tárban, akkor elvégzi a beillesztést is.

Ebből természetesen az is következik, hogy a nem számjeggyel kezdődő sorokat a BASIC mindenképpen parancsként (ill. közvetlen végrehajtású utasításként) próbálja értelmezni.

Az utasítások tehát — bizonyos szempont szerint — felfoghatók úgy, mint a program elemei, és fordítva; a program tekinthető úgy, mint utasítások sorozata. Ez a szempont azonban meglehetősen BASIC-centrikus, így a továbbiakban nem fogjuk használni.

3.3. KULCSSZÓCSOPORTOK

„Ha a madzagnak van egy vége, akkor másik is van neki."

Miksch törvénye

Üzeneteinket sokszor nem fejezhetjük ki egyetlen „szóval", mert pl. az üzenet tartalmazhat több adatot, amelyek viszonyát valahogy le kell írnunk. Ilyenkor az üzenetet bevezető kulcsszó után egyéb kulcsszavak következhetnek; olyanok is, amelyek nem lehetnek bevezető kulcsszavak. Ezek jobbára csak együtt fordulhatnak elő, és természetesen az üzenet szerkezete és formája erősen kötött.

A kulcsszavakat követően megadandó vagy megadható adatokat paramétereknek nevezzük.

3.4. MŰVELETEK, FÜGGVÉNYEK

„Működésképtelen elképzelése mindenkinek van."

Howe törvénye

A számítástechnikában e két fogalmat a matematikából ismert értelemben használjuk. Mindegyikük valamilyen értéket szolgáltat.

A művelet a műveleti szabályok szerint leírt, operandusnak nevezett érték(ek)hez rendeli az eredményül kapott értéket. Megkülönböztetünk egyoperandusú

(pl.: NOT) és kétoperandusú (pl.: összeadás) műveleteket. (Kettőnél több operandus a BASIC-ben nem fordul elő.)

A függvény a zárójelbe írt argumentum(ok)hoz rendeli a függvényértéket. A függvénynek lehet több argumentuma is, ilyenkor azonban ezek sorrendje mindig kötött. A függvények argumentuma(i) és értéke egyaránt lehetnek szöveg- vagy szám-típusúak, de egy-egy függvény esetén mindez szigorúan meghatározott. (Néhány interpreter nem követeli meg, hogy az argumentumot zárójelbe tegyük, de tehetjük.)

4. A PROGRAMOZÁSRÓL

„Ha az építőipar úgy építene, ahogy a programozók készítik programjaikat, az első jöttment harkály lerombolhatná az egész emberi civilizációt."

Weinberg második törvénye

Akinek már futott egy legalább néhánysoros programja, az úgy véli, hogy az első lépést megtette a programozáshoz, és csak a későbbi kudarcai ébresztik rá — szerencsés esetben —, hogy még csak a rajtvonal előtt készülődik.

A BASIC nagyon hamar sikerélményhez juttatja a kezdőt: lenyűgöző dolog átélni, hogy „mondtam valamit a gépnek az ő nyelvén, és megértette, sőt végre is hajtotta azt". Ez a sikerélmény egyesek számára ösztönzést jelent a további tanulásra, mások viszont — éppen ennek alapján — feleslegesnek ítélik a tanulást.

Őszintén el kell mondani azt is, hogy a „heurisztikus" módszer kezdetben hatékonyabbnak bizonyulhat. Azonban igen hamar kiderül róla, hogy a fejlődést, a továbblépést már középtávon is gátolja, néha egyszerűen lehetetlenné teszi. Ezért van szükség a BASIC előtt a programozással megismerkedni.

Elsőként tisztázzuk a program fogalmát! Ezt ugyanis kétféle értelemben is szoktuk használni, de — szerencsére — ez soha nem okoz gondot, ugyanis a tisztességesen megfogalmazott szövegkörnyezet megszünteti a kétértelműséget.

Tágabb értelemben *programnak nevezzük egy* (végrehajtandó) *folyamat tervét.* Ilyen értelemben szoktuk a „családi programot", a „hétvégi programot" vagy a „rádió programját" emlegetni. Az sem zavar bennünket — helyesen —, hogy egy ilyen folyamat valamely részfolyamatára is programként utal valaki, valahogy így: „az esti program tévénézés lesz". A program tehát a folyamat valamilyen formájú leírását jelenti.

Szűkebb értelemben *programon egy folyamat adott nyelven megfogalmazott* (kódrendszerrel leírt) *tervét értjük.* Ilyenkor a beszédben inkább „BASIC programot" vagy „gépi kódú programot" mondunk, esetleg egy korábbi utalás igazítja el a beszélő partnert.

Természetesen a továbbiakban is ragaszkodni fogunk a program ezen kétféle értelmezéséhez, értelmetlen lenne ugyanis dacolni a köznyelvi beidegződéssel, de a fogalom használata során nem engedhetünk meg semmiféle köznyelvi pontatlanságot. Programon tehát mindig folyamattervet értünk, és sohasem utasítássorozatot. Ez utóbbit — ha szükségünk lesz rá — legfeljebb utasításcsoportként fogjuk még emlegetni.

Az elmondottak szerint a programozás tehát folyamatterv-készítés vagy — rövidebben — folyamattervezés.

4.1. A PROGRAMOZÁS FÁZISAI

„A legegyszerűbb gondolatokat fejezik ki a legbonyolultabban."

Malek törvénye

A programozás első fázisa a célkitűzés. Itt kell megfogalmazni a megoldandó feladatot.

A második fázis a megoldási folyamat megtervezése. Ezt azonban gyakran nem gépre vihető formában írták le, ezért szükséges egy további átalakítás.

A harmadik fázis a kódolás, amikor a már kész folyamattervet a gép kódrendszerére „lefordítva" írjuk le, azaz elkészítjük a megfelelő nyelvű programot.

Basic program írásakor a folyamatos párbeszédes üzemmód miatt a fenti fázisok erősen összemosódhatnak. Néhányan ugyanis úgy kezdik meg a kódolást, hogy még az elérendő célok sem tisztázottak, a megoldás módjáról tehát szó sem eshetett. Ez természetesen nagyon rossz programozói gyakorlat, hiszen az így létrejövő program — a rengeteg javítás és próbálgatás miatt — rendkívül kusza, áttekinthetetlen lesz. Az ilyen programokban gyakran vannak soha végre nem hajtódó részek, és persze — a ritkán végrehajtódó részekben — olyan hibák, amelyek utálatos meglepetéseket okozhatnak.

4.2. A CÉLKITŰZÉS, A FELADAT MEGFOGALMAZÁSA

„Tévedni emberi dolog, de isteni érzés."

Mae West megfigyelése

Mint már említettük, a programkészítés első fázisa a program céljának egyértelmű, szabatos leírása, annak megfogalmazása, hogy miket várunk a készülő programtól. Ehhez a legelső lépés három kérdés egyértelmű és pontos megválaszolása.

Először is tisztázzuk, hogy milyen bemenő adatokat kell a programnak feldolgoznia, ill. kezelnie, ezek az adatok milyen logikai szerkezetűek. Mik tekinthetők a feldolgozás során adatelemeknek, azaz olyan adatnak, aminek a program már nem vizsgálja a szerkezetét. Meg kell becsülni az adatok mennyiségét, és ismerni kell a forrásukat is.

Másodszor meg kell adni a kimenő adatokat, ezek logikai szerkezetét és kiviteli formátumát. Illik becslést adni a kimenő adatok mennyiségére is, és az adathordozót is érdemes megjelölni. Ha az adat egy másik berendezésre kerül, akkor ezt természetesen itt kell tisztázni (pl. folyamatirányítás, mérésvezérlés esetén).

Harmadszor azt kell tisztázni, hogy a bemenő adatokból hogyan állíthatók elő a kimenő adatok. Le kell írni — természetesen szavakkal —, hogy a számítógépprogramnak milyen bemenet—kimenet átalakítást (leképezést) kell végrehajtania. Általában ilyenkor derülnek ki az első hibák: rájövünk, hogy valamelyik kimenő adat a bemenőkből nem állítható elő, de az is előfordulhat, hogy egy adatot nem használunk fel. (Sajnos ez az utóbbi hiba olykor nehezen ismerhető fel.)

Ha az első kérdésre való többszöri visszatérés után végre sikerül mindhárom kérdésre (bemenet, kimenet, átalakítás) megnyugtató választ adni, akkor reménykedhetünk abban, hogy az algoritmust sikerül majd elkészíteni.

4.3. A MEGOLDÁS ALGORITMUSA

> *„Sose törd a fejed olyasmin, amit biztosan lehet tudni!"*
>
> *De Never elkallódott törvénye*

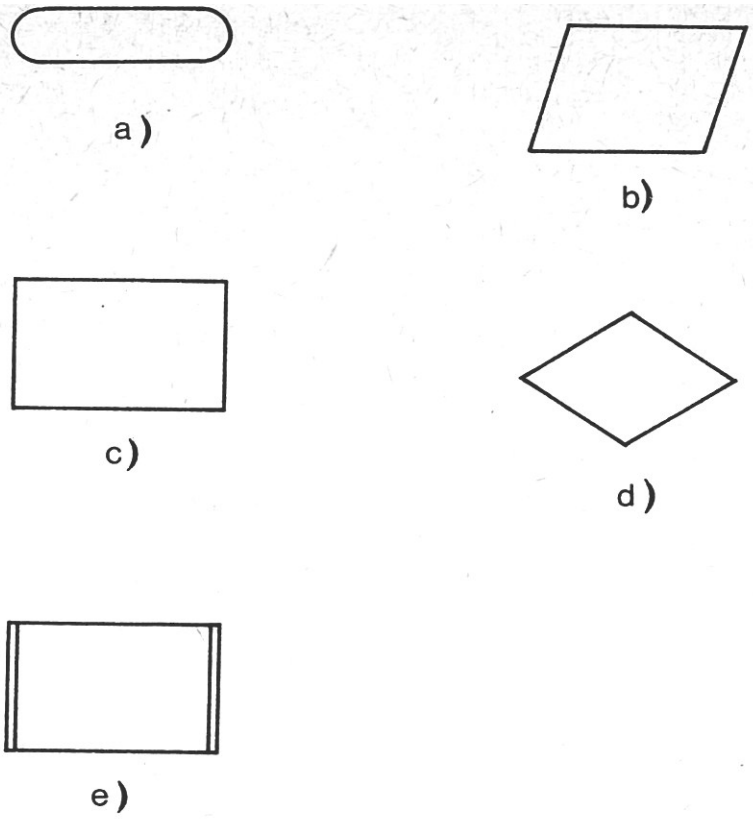
Amikor tisztáztuk, hogy milyen leképezést kell a bemenő adatokon végrehajtani, még nem foglalkoztunk azzal a kérdéssel, hogy ezt miként kell majd a feldolgozás során elvégezni. Igazából eddig nem jelentett különösebb nehézséget a folyamatok leírása: a beszélt emberi nyelvek is lehetővé teszik ezt. A leképezés folyamatának első leírására jó, ha mi is ezt a jelrendszert használjuk, ugyanis ez eléggé közérthető, így ennek segítségével programunkat más is megértheti, programhiba esetén a javításban is segíthet. A folyamat ezen megfogalmazása — természetesen — még csak vázlatos, de a további feladat éppen ennek részletezése lesz.

Könnyű belátni, hogy így a részletezés előbb-utóbb olyan terjedelmes leírást fog eredményezni, amit már képtelenek leszünk áttekinteni. Szükségesnek látszik tehát egy olyan jelrendszert bevezetni, amivel jól áttekinthetően leírhatunk bonyolult folyamatkapcsolatokat, ugyanakkor a részletezést lényegében korlátlan mélységig folytathatjuk. Erre a célra elég jól használható, és a szakirodalomban is elterjedt a folyamatábra, vagy — más néven — a blokkdiagram. Bár a folyamatábra igen jól használható, mégsem jelenti nehézségeink általános érvényű megoldását. Használatát tehát az indokolja, hogy nem terjedt még el ennél jobb jelrendszer, ha egyáltalán létezik jobb.

A következőkben megállapodunk a folyamatábra részeinek jelentésében. (Erre azért van szükség, hogy közös nyelvet beszéljünk, azaz az Olvasó megérthesse a közölt gondolatokat, gondolatmeneteket.) A megállapodásban fő szempont a minimalizálás, azaz a lehető legkevesebb elem bevezetése. Mindössze egyetlen lehetőséget hagyunk a bővítésre, ez ugyan korlátlanul folytatható, de szigorú szabályok által kötött. Végül valljuk be töredelmesen azt is, hogy jelrendszerünk egyetemességén belül megtűrünk egy viszonylag erős „BASIC-központúságot", vagyis az írások (szigorúan nem rögzített) szabályaiban a BASIC formalizmusát tekintjük irányadónak. Ennek haszna a kódolás során lesz érezhető, míg az egyetemesség előnyét csak a következő programozási nyelv tanulásakor élvezheti majd az Olvasó.

A folyamatábrák tevékenységeket jelölő blokkokból és az ezeket összekötő, így a tevékenységek sorrendjét kijelölő vonalakból állnak. Az összeköttetések irányát csak akkor szokás feltüntetni, ha az egyébként nem nyilvánvaló, vagy az áttekinthetőséget javítja. A tevékenységeket öt különböző blokkal jelöljük.

Határoló blokk (1*a* ábra). A START- vagy STOP blokk. A START jelű blokkból egyetlen vonal indul ki, míg a folyamat végét jelző STOP-blokkba csak legalább egy vonal érkezik. Minden folyamatábra tartalmaz legalább egy START-blokkot. (Egy folyamatterv tartalmazhat több belépési pontot, azaz olyan helyet, ahol a folyamat végrehajtása kezdeményezhető. Ilyen lehetőségeket biztosít pl. az automata mosógép programkapcsolója. A folyamatnak lehet több záróblokkja is, azonban ezek



1. ábra

mindig helyettesíthetők egyetlen blokkal. Másrészt az is elképzelhető, hogy egy folyamatnak nincs záróblokkja, azaz a folyamat végtelen. Ezt csak kivételes esetekben, pl. játékprogramokban szabad megengedni.)

Be-, ill. kiviteli blokk (1*b* ábra). A folyamat az ilyen blokkoknál adatot vár vagy adatot szolgáltat. Az egyszerűbb folyamatábrákon nincs szükség az adatforgalmat lebonyolító eszköz megjelölésére, pl. azért sem, mert ezek általában csak kevés eszközt használnak. A blokk megértéséhez tisztázni kell a változó fogalmát.

A folyamatok leírásakor az adatokat névvel látjuk el, így ezeket a neveket használhatjuk az adatok kijelölésére. A bevezetett nevek bármely pillanatban jelentenek valamilyen adatot, ha tehát egy névre hivatkozunk, akkor tulajdonképpen ezt az adatot jelöljük meg. Ezt úgy fejezzük ki, hogy a nevekhez, az ún. (változó) azonosítókhoz mindig tartozik aktuális érték, ami változhat. Innen ered az elnevezésük is. (A számítógépben ezt úgy oldják meg, hogy a változónév egy meghatározott tárterületet jelöl ki, és az általa meghatározott adatot itt tárolják. Erről később részletesebben is lesz szó.)

Bevitel esetén az adatot mindig valamilyen változónévhez rendeljük, amit röviden úgy mondunk, hogy a változóba betöltjük a szóban forgó adatot.

Kivitel esetén is általában a változók tartalmát másoljuk át a megfelelő egységbe, de néha az is előfordul, hogy rögzített adatot, pl. szöveget kell kivinni a képernyőre.

Általános műveleti blokk (1*c* ábra). A legszigorúbb esetben csak értékadásokat

engedünk meg ebben a blokkban, és esetleg még a matematikai kifejezés bonyolultságát is korlátozzuk. Ez magasszintű programozási nyelv esetén nem szükséges, sőt kimondottan kellemetlen lenne. Itt érdemes mindazt megengednünk, amit a BASIC is lehetővé tesz.

Természetesen a fentieken kívül egyéb műveleteket is megengedhetünk. Ezek közül két változó tartalmának cseréjét kell megemlíteni, ugyanis ez a művelet bizonyos típusú folyamatokban elég gyakori.

Elágazási blokk (1*d* ábra). A folyamat valamilyen feltételtől függő elágaztatását végző blokk. Ezt szokás döntési blokknak is nevezni. A blokkban egy feltétel áll, amelynek logikai értékétől függően (igaz vagy hamis) a megfelelő kiinduló vonal irányában folytatódik a végrehajtás.

Jegyezzük meg, hogy ez az egyetlen olyan blokk, amiből egynél több vonal indul ki (pontosan kettő). Azt általában megengedik, hogy egy blokkba (kivéve a START-blokkot) több vonal érkezzék, azonban szokás ezeket — és az áttekinthetőség miatt általában célszerű is — a blokk előtt egyesíteni, és a blokkba való belépést már csak egyetlen vonallal jelölni. Az elágazási blokk esetén ez követelmény.

Eljárásblokk (1*e* ábra). Ezzel egy másutt részletezett, egyetlen START- és egyetlen STOP-blokkal határolt folyamatra hivatkozhatunk. A blokkba a folyamat nevét írjuk.

Itt értünk el az algoritmus fogalmához. Mi a későbbiekben olyan eljárást (folyamattervet) fogunk algoritmusnak nevezni, amely egyértelmű, tehát egyetlen belépési pontja van, amely minden tevékenység után egyértelműen meghatározza a rákövetkezőt, amely véges sok lépés után befejeződik, és az eredménye független a végrehajtótól. (Ennél pontosabb meghatározásra nem lesz szükségünk.)

A feladat megoldásának algoritmusát tehát leggyakrabban folyamatábrával adjuk meg, amit a szükséges mélységig részletezünk. A részletezés akár egyetlen rajzon, akár egy-egy részfolyamat különálló megrajzolásával történhet.

4.4. A PROGRAM MODULJAI

> *„Egy gramm alkalmazás felér egy tonna elvonatkoztatással."*
>
> *Booker törvénye*

Aki már néhány programot készített, az tudja, hogy számos visszatérő feladat adódik, különösen hasonló témájú programok esetében. Az ilyen részfeladatok ismételt megoldása roppant gazdaságtalan lenne. Ha azt is figyelembe vesszük, hogy ezek egy részére már igen kifinomult eljárások készültek, akkor nyilvánvaló, hogy érdemes velük megismerkedni, mintsem újra „kitalálni" egy sokkal gyengébb változatukat. A következőkben néhány ilyen modult, ill. rutint mutatunk be.

Nem a tévhit — miszerint a számítógép számító gép — erősítése miatt választjuk elsőként két szám legnagyobb közös osztójának a meghatározását. Az ok inkább az, hogy ezt a feladatot szinte mindenki ismeri, és így figyelmünket az eljárásra és leírására fordíthatjuk *.

Az algoritmusban felhasználjuk, hogy két szám *legnagyobb közös osztója* osztója a két szám különbségének is. Az eljárás elve röviden a következő: addig csökkentjük kivonásokkal a két szám közül a nagyobbikat, míg a két szám egyenlő lesz. A következő kivonás eredménye már nulla, és innen ismerhető fel az algoritmus vége. Nem nehéz megmutatni, hogy az utolsó nem nulla különbség épp a kiindulási két pozitív szám legnagyobb közös osztója**.

Ezek után nézzük a folyamatábrát! Tegyük fel, hogy a vizsgálandó két egész szám az U és a V változóban van, értékük nem negatív. (A feltételezés azért jogos, mert az eljárás kezdeményezésekor ezeket könnyen biztosíthatjuk, viszont a vizsgálatokat és az átalakításokat itt végeztetve a programunk sokkal kevésbé lenne általános.)

Megvizsgáljuk, hogy V értéke nulla-e. Ha igen, akkor a legnagyobb közös osztó éppen U. (Ez még akkor is igaz, ha U értéke is nulla!) Az U tartalmát tehát áttöltjük V-be, és az eljárásnak vége. Különben továbbmegyünk.

Megvizsgáljuk, hogy U kisebb-e, mint V. Ha igen, akkor V értékét csökkentjük U értékével, különben U értékét csökkentjük V értékével.

Ha U értéke nulla lett, akkor V az eredményt tartalmazza és vége az eljárásnak, különben vissza kell térni a második vizsgálatra.

Megjegyezzük, hogy a legnagyobb közös osztó előállítására más eljárások is léteznek, azonban ez az egyik legegyszerűbb és legkönnyebben gépre vihető algoritmus. Nem foglalkoztunk az algoritmus részletes vizsgálatával, de ezt — különösen ha magunk készítjük — ne mulasszuk el, ugyanis előfordulhat, hogy az eljárásunk nem ér véget, vagy olyan sokáig tart, hogy kivárhatatlan. Az Olvasóra bízzuk annak átgondolását, hogy eljárásunk véges sok lépés után befejeződik.

A következőkben három adatkezelő algoritmust fogunk megismerni. Közös tárgyalásukat az indokolja, hogy ugyanolyan logikai szerkezetű adatcsoportok kezeléséről van szó, másrészt ezek olyan alapvető műveletek, amelyeket érdemes átgondolni egyszer.

Az adatcsoportunk ismert számú, vagy ismert és rögzített korlátnál nem nagyobb számú, azonos típusú adatból áll. Ilyen pl. egy névsor vagy egy mérési sorozat eredményei. Az adatokat ilyen esetben egy (közös) névvel és a sorszámukkal érdemes

* A definíció szerint azt mondjuk, hogy az a egész szám osztója a b egész számnak, ha létezik olyan c egész szám, hogy $b = c \cdot a$. Ebből adódik, hogy ha a osztója b-nek, és b pozitív, akkor b nem kisebb a-nál.

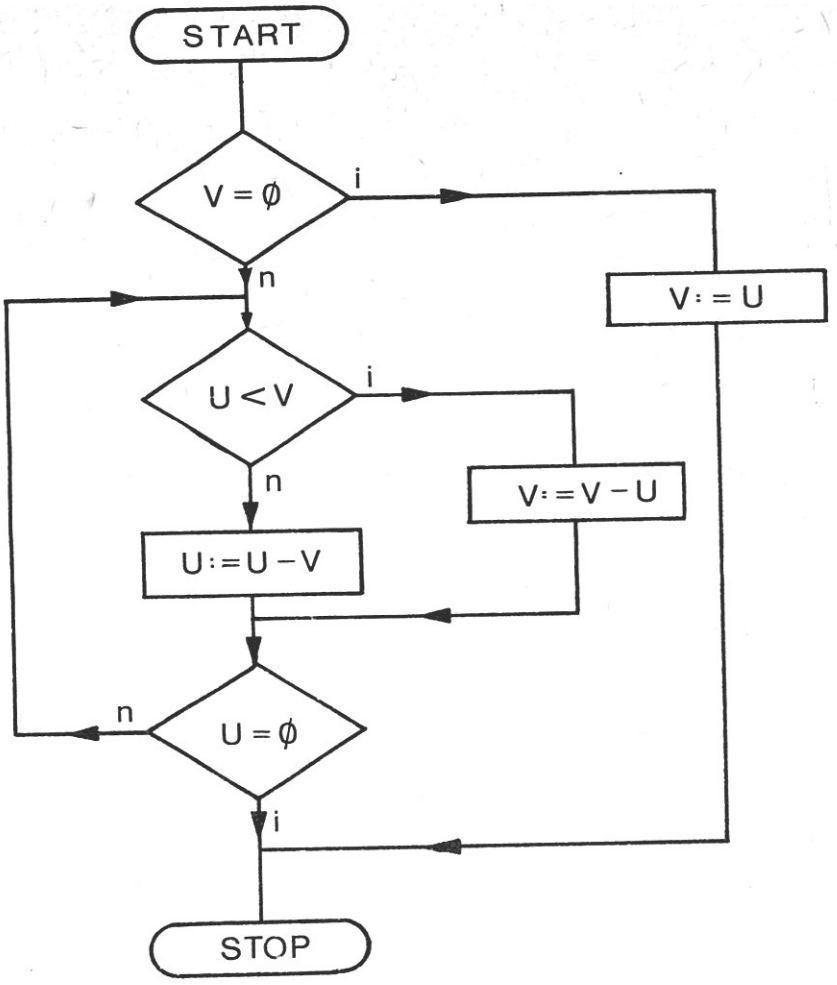
Bizonyítás: ha a nem pozitív, akkor az állítás nyilvánvaló. Pozitív a esetén $b = c \cdot a$ miatt c is pozitív, így legalább *1*, tehát a jobb oldali szorzat a-nál nem kisebb.

A legnagyobb közös osztó szokásos definíciója a következő: az a és b egész számok legnagyobb közös osztóján értjük azt az egész számot, aminek minden közös osztó osztója.

Könnyű látni, hogy a fenti definíció szerint a legnagyobb közös osztó képzése nem egyértelmű, hiszen ha egy egész szám legnagyobb közös osztó, akkor az ellentettje is az. Ezt a nehézséget úgy szokás áthidalni, hogy a legnagyobb közös osztót nem negatívnak választjuk.

Ennek alapján az elnevezés eredete a fentebb bizonyított megjegyzés figyelembevételével érthető. A gondot az okozza, hogy a definíció szerint a legnagyobb közös osztó 0, ha mindkét szám 0, pedig ekkor minden egész szám közös osztó, ezek között viszont nincs is legnagyobb. Éppen ez a furcsaság indokolja a fenti definíciókat!

** *Bizonyítás:* ha a d egész szám osztója a-nak és b-nek, akkor $a = k \cdot d$ és $b = l \cdot d$, így $a - b = (k - l) \cdot d$, tehát d a különbségnek is osztója. Minthogy ez minden közös osztóra érvényes, és a folyamat egymás utáni lépéseire alkalmazható, így azt kapjuk, hogy az utolsó nem nulla különbségnek minden közös osztó osztója. Másrészt az is nyilvánvaló, hogy ez a különbség az összes előző különbségnek osztója, így a kiinduló számoknak is, tehát közös osztó.



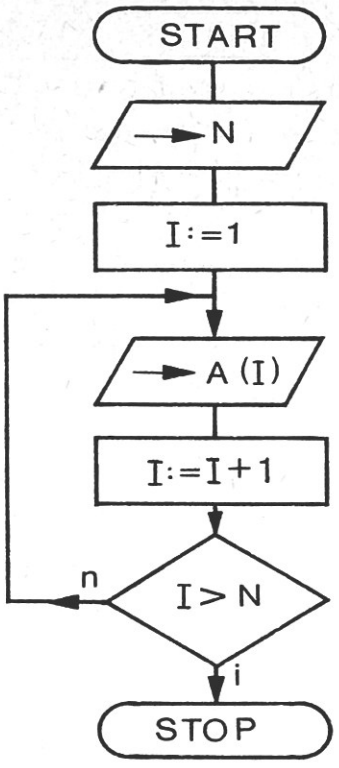
2. ábra

megjelölni. A sorszámot indexnek, az egész adatcsoportot tömbnek, az adatok közös nevét tömbnévnek, míg a tömb egy-egy adatát tömbelemnek nevezzük. Egy tömbelemet úgy adunk meg, hogy a tömbnév után zárójelbe írjuk az elem indexét. A tömb elemeinek számát a tömb méretének mondjuk. Elképzelhető, hogy egy adatcsoport több indexszel jellemezhető. Pl. egy táblázat adatait úgy azonosíthatjuk, ha megadjuk, hogy hányadik sorban és hányadik oszlopban van. Az ilyen tömböket többdimenziós tömböknek nevezzük. A tömb dimenziószáma az indexek számát jelenti, míg az egyes indexek felső korlátait dimenziónkénti méretnek szokás nevezni.

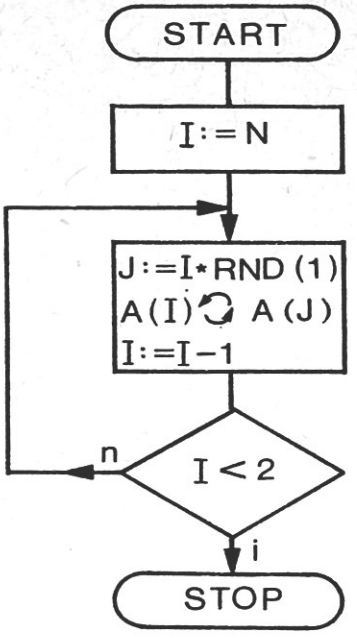
Elsőként egy tömb bevitelére szolgáló eljárást (3*a* ábra) nézzünk! Így olvastathatunk be pl. egy osztálynévsort is. Minthogy a folyamatábra magáért beszél, így elég csak néhány megjegyzés. Az algoritmus csak akkor használható, ha előre ismert az adatok (pl. nevek) száma. Akármilyen adatszámot mondunk, egy adatot mindenképpen beolvas az algoritmus, hiszen a vizsgálat csak a beolvasás után történik. Könnyű látni, hogy a tömb tartalmának kiírása ugyanígy történhet, ilyenkor azonban már tudjuk az adatok számát, ezt tehát nem kell beolvastatni, és természetesen a tömbelem beolvasása helyett ennek kiírását kell végezni.

Feladat. Készítsük el azt a folyamatábrát, amivel egy osztály nyilvántartását olvashatjuk be! Tanulóként a következő adatokat kérjük: név, anyja neve, születési hely, személyi szám és lakcím. Az adatokat egy kétindexes tömbbe helyezzük el!

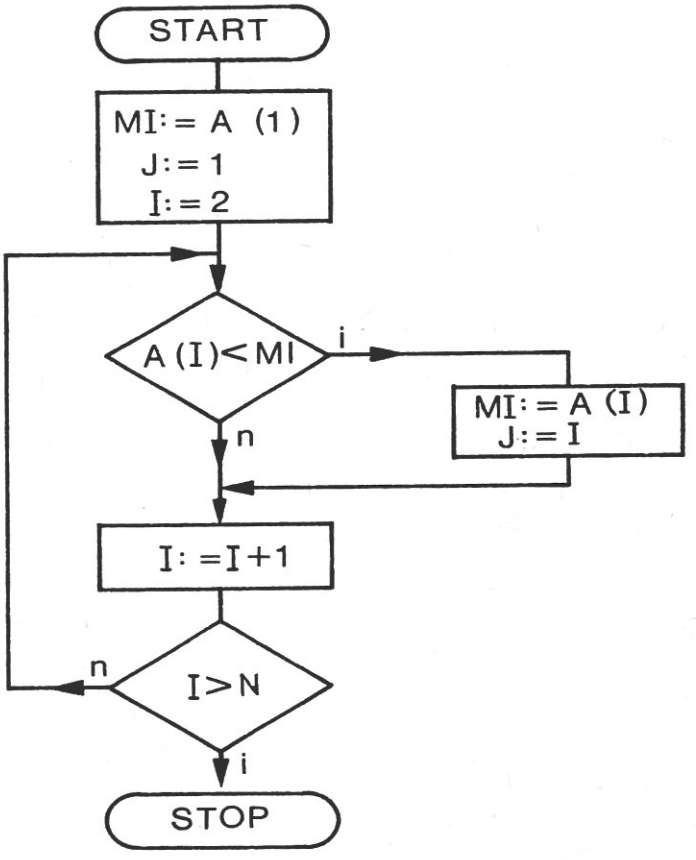
Második példánk egy tömb keverése. Ilyen feladat pl. egy sportverseny résztvevőinek a sorrendjét kisorsolni, de lehet egy kártyacsomag összekeverése is. A szemléletesség kedvéért az elvet is kártyával mutatjuk be.


START
N
I:=1
A (I)
I:=I+1
I > N
n
i
STOP

a)


START
I:=N
J:=I*RND (1)
A(I) A(J)
I:=I-1
I < 2
n
i
STOP

b)


START
MI:= A (1)
J:= 1
I:= 2
A (I)< MI
i
n
MI:= A (I)
J:= I
I:=I+1
I>N
n
i
STOP

c)

3. ábra

Válasszuk ki a csomag egy kártyáját és ezt cseréljük ki a legfelsővel! A továbbiakban ezt már helyben hagyjuk.

A maradékból válasszunk ki egy lapot és ezt cseréljük ki e maradék legfelső lapjával! Ezt addig ismételjük, amíg egy lap marad (3*b* ábra)!

A folyamatábra két műveletéről kell szólni. Az RND (I) — a BASIC-ből átvett jelölés — olyan függvényt jelent, amely 1 és I közötti véletlen egész számot állít elő. Így „soroljuk" a cserélendő adat sorszámát. A következő művelet a megfelelő két elem cseréje, amit a továbbiakban is megengedünk, bár a BASIC nem teszi lehetővé ilyen egyszerűen a cserét.

Feladat. Készítsünk eljárást, amely lottószámokat sorol! (Töltsük fel a tömböt az első 90 természetes számmal, majd keverjük a tömböt amíg szükséges!) A kisorsolt számokat írassuk ki!

A következő példánkban egy adatsor szélső elemeinek keresésére adunk egy eljárást; feladatunkban ez egy névsor abécé szerinti első vagy utolsó elemét adja. Az eljárás lényege röviden a következő: egy-egy változóba töltjük az első elem értékét és ennek indexét. Ezután elindulunk a második elemtől, és ha a tároltnál kisebb (előbb álló) elemet találunk, akkor ezt és ennek indexét tároljuk, és folytatjuk az ejárást addig, amíg az adatsor végére nem érünk. Így az eljárás végén a legelső adatot és annak indexét kapjuk. Az egyenlőtlenség irányát megfordítva az eljárás a legnagyobb (utolsó) elemet és az indexét szolgáltatja (3*c* ábra).

Feladat. Készítsünk másik eljárást, amelyik az előző feladatban kisorsolt lottószámokat növekvő sorrendben írja ki! (Megjegyezzük, hogy a már kiírt számra nincs szükségünk, így tehát helyére egy a táblázatban biztosan nem szereplő számot, pl. kilencvenkilencet írhatunk.)

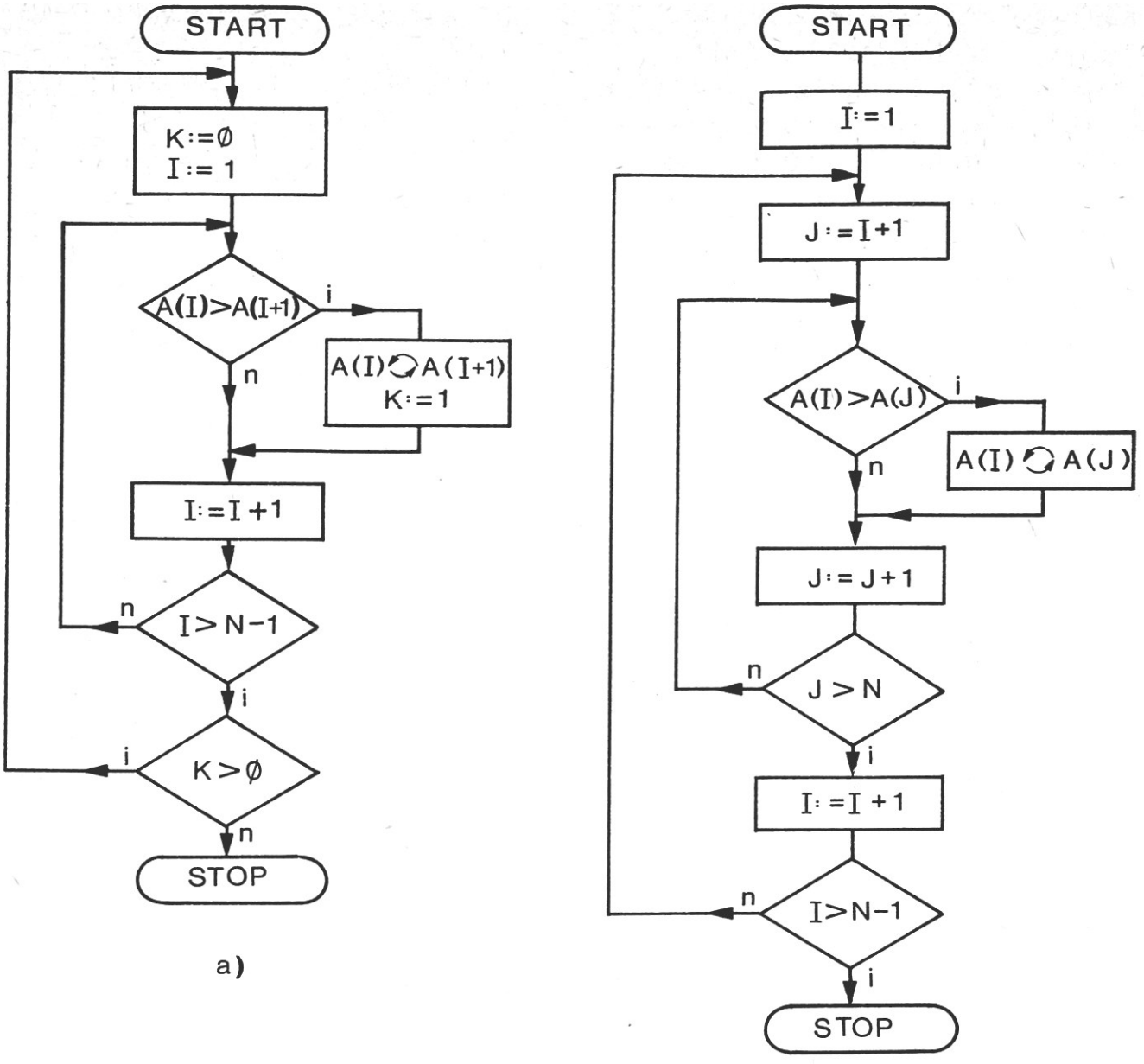
A következőkben két rendező algoritmussal foglalkozunk. Kiválasztásuk fő szempontja az volt, hogy könnyen érthetők és esetleg módosíthatók is. Ennek viszont az az ára, hogy mindkét eljárás lassú: a végrehajtási idő az elemek számának négyzetével arányos.

A *szomszédos elemek cseréjével történő rendezés* (4*a* ábra) elve a következő: az első elemtől indulva minden elemet összehasonlítunk az őt követő elemmel. Ha sorrendjük jó, akkor továbbmegyünk, különben felcseréljük őket, és feljegyezzük, hogy csere történt. Mindezt az utolsó két elemig végrehajtjuk. Ha egy ilyen összehasonlítás-sorozat során már nem kellett cserélnünk, akkor az eljárás véget ért, hiszen ha a szomszédos elemek sorrendje helyes, akkor az egész adathalmaz rendezett.

Érdemes megjegyezni, hogy ez az eljárás a már eleve rendezett tömbön mindössze egyszer fut végig. Azt is könnyű látni, hogy — sajnos — a rendezés végén egyszer feleslegesen fut végig, ugyanis ekkor veszi észre, hogy már nincs teendője, az eljárás tehát befejeződött.

Vegyük észre, hogy az első összehasonlítás-sorozat után a tömb utolsó eleme biztosan a helyére kerül, a második után az utolsó előtti is, és így tovább! Ebből két dolog következik. A többszöri végrehajtások során mindig eggyel több biztosan felesleges lépést teszünk, ez az eljárás hatékonyságát lényegesen rontja. Másrészt az is nyilvánvaló, hogy az utolsó elemtől kezdve mindig eggyel több elem biztosan a helyére kerül, így az eljárás véges, sok lépés után befejeződik.

A buborékos rendezés tulajdonképpen az előző követelmény szellemes kihasználásával működik. Működési elve vázlatosan a következő: az első elemet összehasonlítja az összes többivel. Ha az az elsőnél nem kisebb, akkor tovább lép, különben kicseréli őket és tovább lép. Ha a tömb végére ért, akkor az első elem biztosan



4. ábra

a helyére került, és az eljárást a második elemre is végrehajtjuk, és így tovább az utolsó előtti elemig. (Vegyük észre, hogy ha egy adathalmaz elemei az utolsó előtti elemig már a helyükön vannak, akkor az utolsónak is a helyére kellett kerülnie! (4*b* ábra.)

Nyilvánvaló, hogy az összehasonlítások relációjelének megfordításával a rendezés eredménye fordított lesz, tehát egyaránt használhatók növekvő és csökkenő rendezéshez az eljárások. Minthogy a magasszintű programozási nyelvek lehetővé teszik két szöveg összehasonlítását is, így a rendezési algoritmusok egyszerűen használhatók szövegek abécé-rendbe szedéséhez is. Az összehasonlításban az ábécé szerint előbb álló szöveg a kisebb, így az abécé-rendben lévő szöveg növekvő rendezettségű.

Feladat. Készítsünk rendezési algoritmust, amely a legkisebb elem kikeresésével végzi a rendezést: kikeresi a tömb legkisebb elemét és kicseréli az első elemmel, majd a tömb maradék részével megismétli az eljárást mindaddig, amíg az utolsó előtti elem is a helyére kerül!

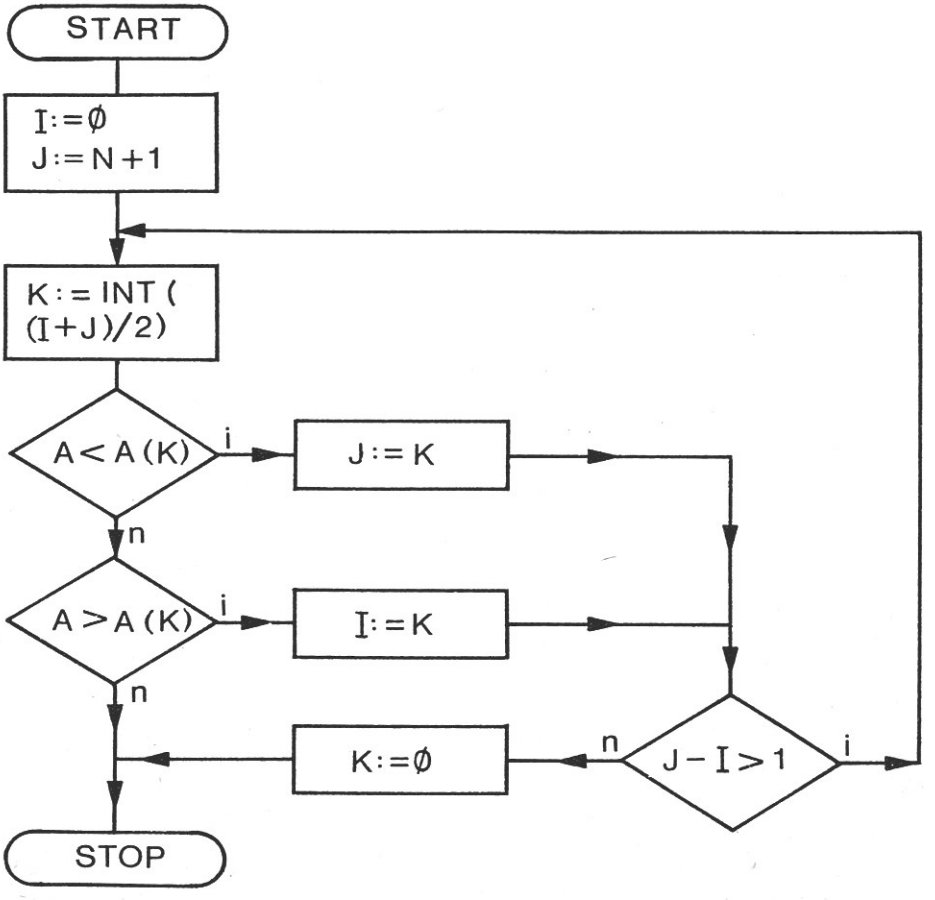
Befejezésül egy kereső eljárást nézünk meg, amely *rendezett* tömbben való kere-

sésre szolgál. A *logaritmusos keresés* elve a következő: megnézzük a középső elemet azt vizsgálva, hogy a keresett elem előtte van-e. Ha igen, akkor a továbbiakban a tömb első felével folytatjuk ugyanígy a vizsgálatot. Ha nem, akkor megvizsgáljuk, hogy a második felében van-e. Ha igen, akkor a tömb második felével folytatjuk ugyanígy a vizsgálatot, ha nem, akkor megtaláltuk a keresett elemet. Az eljárást addig folytatjuk, amíg vagy megtaláljuk a keresett elemet, vagy már két szomszédos elemet is megvizsgáltunk. (Ezek egyike a keresett előtt, másika pedig utána lenne a tömbben.) Természetesen ez utóbbi esetben a keresett elem nem fordul elő az adatok között.

Figyeljük meg, hogy kereséskor mindig megfelezzük a tömböt, így a keresés rendkívül hatékony. (Megmutatható, hogy közelítőleg az elemek számának kettes alapú logaritmusával egyenlő egy tetszőleges elem megtalálásához szükséges átlagos lépésszám.)

Azt is könnyű látni, hogy az algoritmus olyankor is értékes adatot szolgáltat, amikor a keresett elem nem fordul elő a tömbben: ilyenkor az alsó és a felső határ által kijelölt elemek közé kell a szóban forgó elemet beszúrni ahhoz, hogy a tömb rendezett maradjon.

Megjegyezzük azonban, hogy némi nehézséget okoz, ha a tömbben azonos elemek is vannak, bár a rendező algoritmusaink ilyen esetekben is kifogástalanul működnek. Az 5. ábrán látható algoritmust tehát csak szigorúan növekvő adattömbnél használhatjuk ilyen egyszerűen.



5. ábra

Feladat. Készítsünk olyan algoritmust, amely szigorúan rendezetten viszi be az egyébként tetszőleges sorrendben begépelt adatokat! (Az aktuálisan bevitt adatot a tömb megfelelő helyére szúrja be, ha pedig ilyen adat már van, akkor a beszúrást hagyja el!) Az adatbevitel végét valamilyen speciális adat bevitelével jelezzük!

Feladat. Készítsük el a lineáris keresés folyamatábráját, amely egy (esetleg nem szigorúan) rendezett tömbben a keresett elem első és utolsó előfordulásának indexét szolgáltatja! (Először a tömb elejétől indulva, egyenként vizsgálva az elemeket, megkeressük az első előfordulást, majd addig lépegetünk tovább, amíg ugyanilyen elemek vannak a tömbben. Gondoljuk át, mit érdemes tenni, ha a tömb nem tartalmazza a keresett elemet!)

Az ismertetett és a feladatokban szereplő algoritmusok lehetnek a programjaink ún. moduljai. Olyan logikai és működési egységek, amelyek egy viszonylag összetettebb feladat megoldását végzik. Programozáskor célszerű ilyen modulokból építkezni, mert az ezekből álló program viszonylag kevés meglepetést okoz majd.

4.5. A KÓDOLÁS

> *„Mindent lehet egyszerűbben is csinálni."*
>
> *Iles törvénye*

A program kódolását általában mechanikusan végezhető munkának tartják a szakemberek. Ez azért alakult így, mert a nagy számítógépek a magasszintű nyelven megírt ún. forrásprogramból egy a gép központi egysége számára kódolt, általa igen gyorsan végrehajtható programot állítanak elő. Ez a fordítás csak egyszer történik: a továbbiakban már csak ezt a lefordított programot futtatják, így tehát a kódolás alig befolyásolja a futási időt.

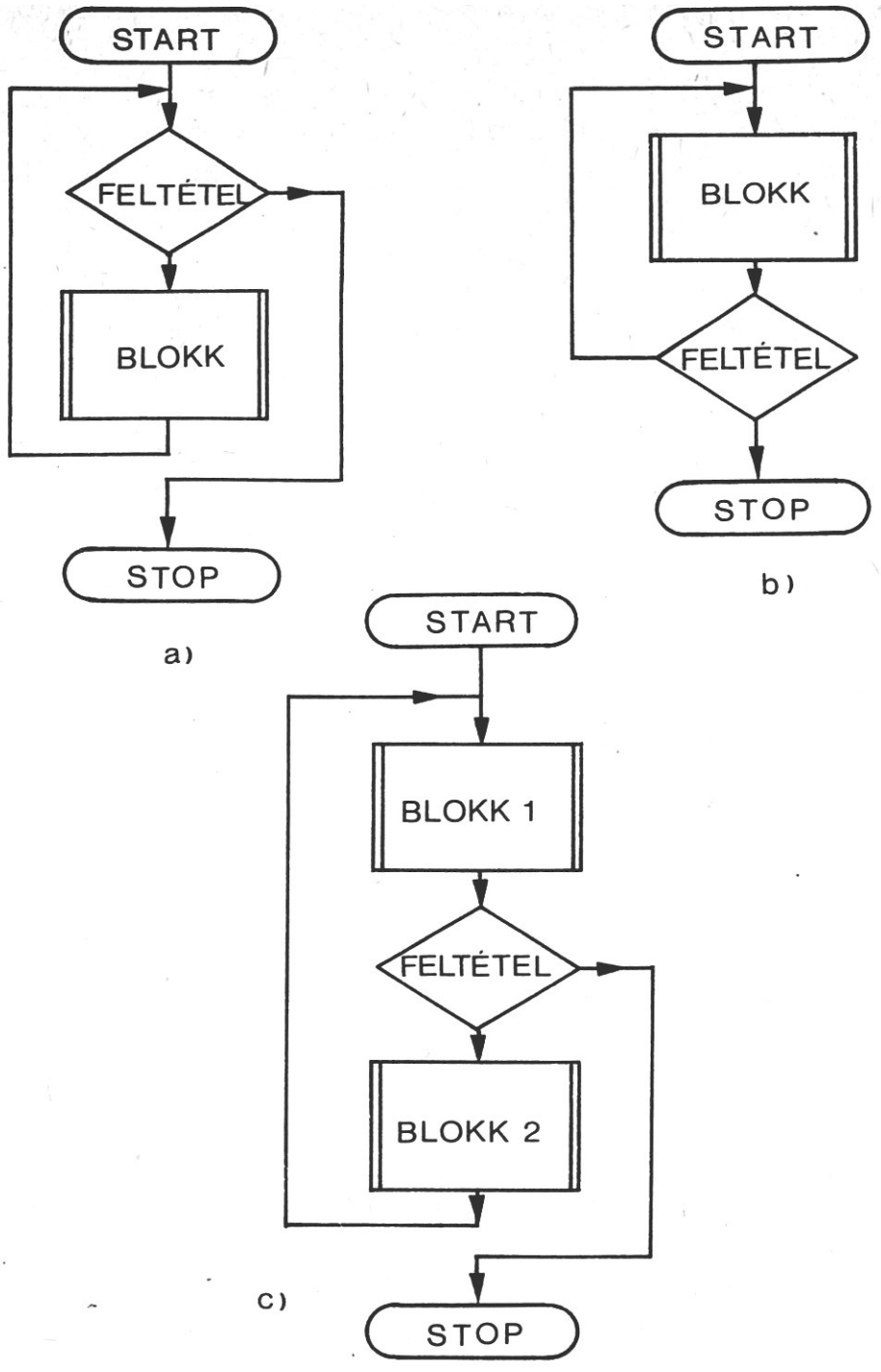
Alapvetően más a helyzet a személyi számítógépekkel. Ezek a magasszintű nyelven megírt programot értelmezik: azaz a fordításhoz hasonló lassúságú művelet során jön létre a kívánt eredmény. Ez tehát tulajdonképpn azt jelenti, hogy többszöri végrehajtáskor az utasítást minden alkalommal mintegy lefordítja a gép.

A fentiek alapján tehát az értelmezővel (interpreter) működő gépek alkalmazásánál sok múlhat a kódoló ügyességén. Érdemes tehát néhány szót szólni a kódolásról, mielőtt még a BASIC sajátosságaival foglalkoznánk.

Az előzőkben tárgyalt algoritmusok folyamatábráiról látható, hogy gyakran van szükség ismétlődő végrehajtásra. Ilyenkor egy műveletcsoportot — esetleg változó paraméterekkel — többször kell végrehajtani. Ezeket a program-, ill. folyamatrészeket huroknak vagy ciklusnak nevezzük.

Egy ciklus szerkezete alapvetően háromféle lehet. A ciklusfeltétel (amely eldönti, hogy a ciklus végrehajtása befejeződik-e) vizsgálható a ciklus utasításai — az ún. ciklusmag — előtt, a ciklusmag után, ill. a ciklusmag belsejében. Bonyolultabb esetben a ciklus folytatása több feltételtől is függhet, ezek vizsgálata a ciklus különböző helyein állhat. (Megjegyezzük, hogy a feltételek között szerepelhet olyan is, amely cikluson belüli ugrásokat eredményez, az ilyeneket azonban a magasszintű programozási nyelvek külön nem támogatják.)

Az *elöltesztelő ciklus* a 6*a* ábrán látható. A feltételtől függően végrehajtódik a



6. ábra

ciklusmagot alkotó BLOKK, vagy továbbmegyünk. A mag végrehajtása után a feltételvizsgálatra térünk vissza. Ha a ciklus végrehajtásának kezdetén a feltétel nem teljesül, akkor a mag egyszer sem hajtódik végre.

A *hátultesztelő ciklus* (6*b* ábra) a magot alkotó BLOKK végrehajtásával kezdődik. Ezután a feltétel értékétől függően visszatér a mag végrehajtására, vagy befejeződik a ciklus. Nyilvánvaló, hogy a hátultesztelő ciklus magja legalább egyszer végrehajtódik.

A *magban tesztelő ciklus* magja két részből áll. Először a mag első részét alkotó BLOKK 1 hajtódik végre, majd a feltételtől függően a ciklus befejeződik, vagy a mag második részét képező BLOKK 2 végrehajtásával folytatódik, majd a ciklusmag elejére kell visszatérni. Az előzők alapján nyilvánvaló, hogy a BLOKK 1 legalább egyszer mindig végrehajtódik, míg a BLOKK 2 esetleg egyszer sem.

Megjegyezzük, hogy üres BLOKK 1-nél elöltesztelő, üres BLOKK 2-nél hátultesztelő ciklust kapunk.

A magasszintű programozási nyelvek ciklusok szervezésére szolgáló utasításokat is tartalmaznak. Fordító esetén ennek különösebb jelentősége nincs, bár a programlista áttekinthetőségét a ciklusutasítások használata jelentősen javítja. Értelmezővel működő gép alkalmazásánál a helyzet elvileg más: a ciklusutasításokkal szervezett ciklusokban a vezérlésátadások gyorsabban hajtódnak végre, mint az ugró utasítások. (L. még később is.) Minthogy az értelmező vezérlésátadáskor (ugrás) általában a program szövegében keres, így érthető, hogy a keresés hatékonysága erősen függ a programszövegtől. (A változókkal hasonló a helyzet, de erről is csak később lesz szó részletesen.)

Az elmondottakból adódik, hogy BASIC értelmezővel futtatott program ügyes kódolásán rendkívül sok múlhat. Egy szerencsétlenül kódolt program futási ideje ugyanis sokszorosa lehet egy szellemesen megoldottének, ez esetleg egy adatfeldolgozás során nem okoz különösebb gondot, de egy párbeszédes program használatát lehetetlenné teszi. Gondoljunk pl. egy mérésvezérlésre, esetleg egy olyan bemutató programra, amely az előadás vagy a tanóra alatt le sem fut. Ha tehát igazán jó, szellemesen használható programokat akarunk írni személyi számítógépünkre, akkor a programkódolásnak is mesterévé kell válnunk.

4.6. HIBAKERESÉS, BELÖVÉS

> *„Ha egyszer összekutyulódott valami, a kijavítására tett minden kísérlet csak rontani fog rajta."*
>
> *Finagle negyedik törvénye*

Ha programunk kódolásával elkészültünk, akkor elérkeztünk a legizgalmasabb és egyben a legnehezebb művelethez. Programunk ekkor már általában működőképes, de bizonyára rengeteg hiba van benne. Szerencsére a formai (nyelvtani) hibák legnagyobb részét a gép kijelzi, de sajnos olyanok is akadhatnak, amelyek a gép számára szándékunktól eltérő módon mégis értelmezhetők. Az ilyen hibák felderítése csaknem reménytelen.

Követhetünk el algoritmikus hibákat is, azaz előfordulhat, hogy az általunk előírt folyamat nem a kívánt adatokat szolgáltatja.

A program futása során adathibák is keletkezhetnek, akár azért, mert rosszul adtunk meg a program írásakor valamilyen állandó adatot, akár azért, mert elfelejtettük valamilyen művelet előtt az adatok helyességét, szerkezetét vizsgálni, és ezért pl. nullával akarunk osztani, vagy a programban meg nem adott adatokra próbálunk hivatkozni.

A hibák lehetőségeinek felsorolása is olyan hosszadalmas, hogy reménytelennek tűnik egy program belövése; a hibák kijavítása. Szerencsére a helyzet nem ennyire rossz.

Először is azzal védekezhetünk a hibák ellen, hogy a felhasznált algoritmusainkat alaposan átgondoljuk, működésüket — különösen szélsőséges feltételek között — ellenőrizzük (teszteljük). Megvizsgáljuk, hogy egymás működését nem zavarhatják-e:

pl. az egyik algoritmus (programmodul) elrontja egy másiknak valamelyik paraméterét, mert közösen használnak egy változót.

A második védekezési mód a gondos kódolás és gépelés. Mindkettőt folyamatosan ellenőrizzük, és begépelés után a programlistát külön is vizsgáljuk meg!

A harmadik segítség nem a megelőzésre vonatkozik. Próbafuttatások során a gép számos hibát kijelez, megjelölve a hiba helyét és jellegét. Mindez felbecsülhetetlen segítséget jelent még akkor is, ha tudjuk, hogy a hiba gyakran nem ott keletkezik, ahol a gép beleütközik.

Végül a már hiba nélkül lefutó program tesztelése segíthet az algoritmikus hibák felderítésében. Ilyenkor ugyanis próbaadatokkal futtatjuk a programunkat, és vizsgáljuk, hogy a kívántnak megfelelő eredményt kapjuk-e.

Ha az első két lehetőséget megfelelően kihasználtuk, akkor az utóbbi kettő nem fog különös meglepetéseket okozni; elkészült programunk megbízhatóan fog működni.

Jegyezzük meg: egy programot sokkal könnyebb jól megírni, mint hibás változatát kijavítani!

5. AMIT A BASIC-RŐL ÉRDEMES TUDNI

„A számítógépprogram nem kívánságaid, hanem utasításaid szerint működik."
Greer harmadik törvénye

A következőkben a BASIC-értelmezők olyan általános tulajdonságairól lesz szó, amelyeket érdemes egy konkrét géppel való ismerkedés előtt áttanulmányozni. Ennek legalább két előnye van. Egyrészt ismereteink „hordozhatók" lesznek, azaz jól el tudjuk különíteni az általános BASIC-tulajdonságokat a konkrét gépi sajátosságoktól. Másrészt az általános tulajdonságok ismeretében tudjuk igazán értékelni — így használni is — a gépünk nyújtotta különleges lehetőségeket.

5.1. A VÁLTOZÓ. VÁLTOZÓTÍPUSOK ÉS TÁROLÁSUK. KIFEJEZÉSEK

„Bármely program a rendelkezésre álló memória végső határáig duzzad."
A számítógépes programozás ötödik törvénye

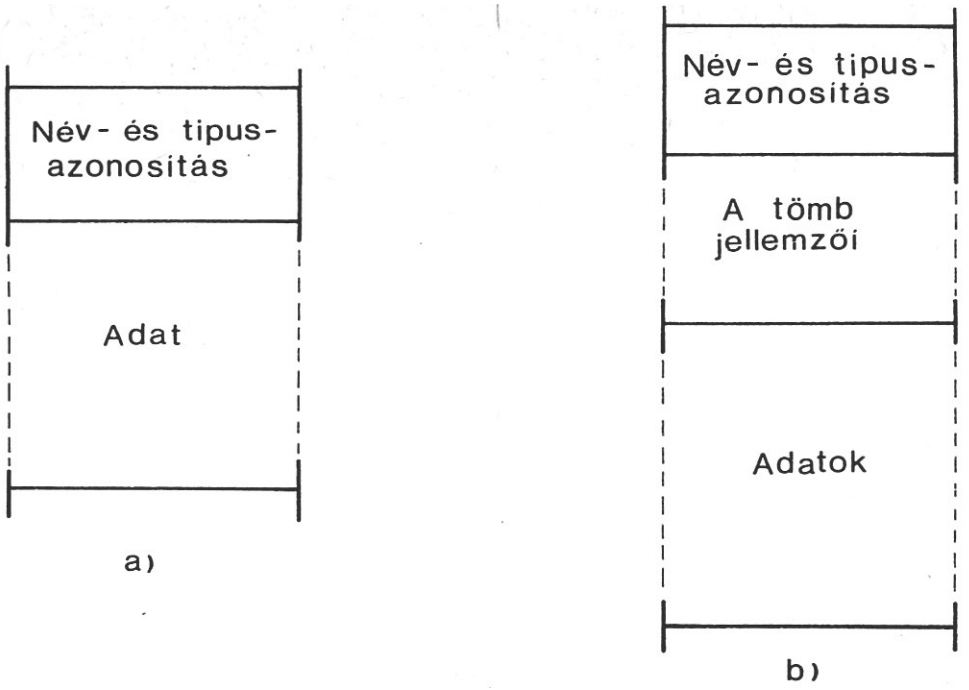
Korábban már volt szó arról, hogy az adatok egy részét névvel látjuk el, így ugyanis az adat konkrét értékének ismerete nélkül is hivatkozhatunk rájuk, éppen a nevükkel. Említettük, hogy a számítógépben a változónév a tár egy területét jelöli ki: a változó aktuális értéke, tehát az az adat, amit a változónév éppen jelent, ezen a tárterületen helyezkedik el.

A magasszintű programozási nyelvek — így a BASIC is — különböző típusú adatok kezelését teszik lehetővé. Ezek egyik csoportja a számtípusú, másik a lánctípusú adatok csoportja.

A BASIC-értelmezők mindegyike kezeli a valós típusú számadatokat. Ezek pontossága általában hattól kilenc tizedes számjegyig terjed. Jó tudni, hogy a legtöbb gép belső számábrázolása bináris, így a kerekítések és a végtelen szakaszos bináris törtek miatt néha furcsa eredményeket kapunk. A decimális számábrázolás előnye, hogy a gép kerekítési hibái megegyeznek az általunk megszokottakkal. A számábrázolás tartománya a belső ábrázolás módjától függően általában $\pm 10^{-38}$-tól $\pm 10^{38}$-ig, vagy $\pm 10^{-127}$-től $\pm 10^{127}$-ig terjed.

A megadott határok természetesen csak tájékoztató jelleggel használhatók, ugyanis az egyes megvalósítások némileg eltérnek.

A komolyabb BASIC-változatok lehetővé teszik egész típusú számok ábrázolását is. Ezeket mindig 16 biten (2 byte) ábrázolják, ábrázolási tartományuk −32768-tól +32767-ig terjed. Az egész típusú szám használata azért célszerű, mert a velük



7. ábra

végzett műveletek gyorsabban hajtódnak végre, ábrázolási hiba nincs és a legtöbb esetben tárolásuk is takarékosabb.

A professzionális célra készült BASIC-értelmezők lehetővé teszik a nagypontosságú számábrázolást is. Ilyenkor jobbára azt a megoldást választják, hogy a valós számok ábrázolásának megfelelő módon ábrázolják ezeket is, azonban nem 4-5, hanem általában 8 byte-ot használnak az adat tárolására.

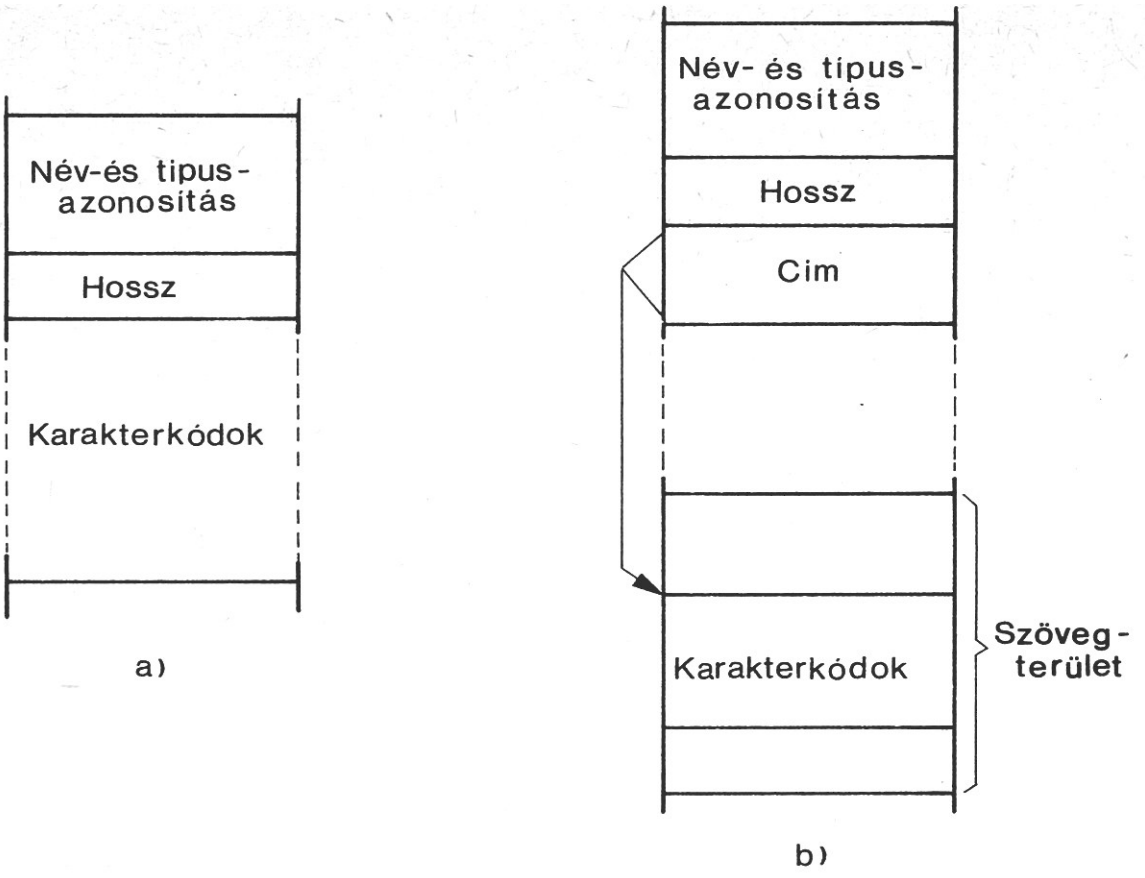
A BASIC-értelmezők általában csak karakterekből álló láncok ábrázolását teszik lehetővé. Az ilyen láncokat sztringnek is, füzérnek is nevezik. Az őket tartalmazó változókat pedig sztringváltozóként vagy szövegváltozóként fogjuk emlegetni. Itt jegyezzük meg, hogy a BASIC-rendszerek általában egy byte-on tárolják a sztringváltozók hosszát, így ez Ø-tól 255-ig változhat, de egyéb megszorítások is lehetnek.

Egy számváltozó tárolását láthatjuk a 7*a* ábrán. Természetes, hogy a változó azonosításához a típusát is meg kell adni, és persze tárolni is kell. A legtöbb interpreter erre néhány byte-ot használ, amelyen a változónév első két karakterét tárolja. Ha a változónév csak egyetlen karaktert tartalmaz, akkor ezt jelzi, de a változónév tárolása ilyenkor is ugyanannyi byte-ot igényel. Ez a megoldás azért előnyös, mert a kötött hosszúságú változónevek közötti keresés lényegesen gyorsabb. A változó azonosítását szolgáló byte-ok után helyezkedik el az adatot leíró megfelelő számú byte.

A számok tárolása tömbváltozóban teljesen hasonló módon történik. Itt a név- és típusazonosító byte-okat előbb a tömb szerkezetét és méreteit leíró adatok követik, és csak ezek után következnek az adatokat tároló byte-ok (7*b* ábra).

Sztringváltozók tárolására kétféle módszer terjedt el. Ezeket a 8. ábrán mutatjuk be.

A 8*a* ábra szerinti tárolás változó hosszúságú szövegeknél okoz nehézségeket. Ennek alapján érthető, hogy néhány BASIC-változat megköveteli a szövegváltozó (maximális) méretének előzetes megadását. Más változatok a nehézséget pl. a változók mozgatásával oldják meg, aminek viszont a körülményesebb cikluskezelés lesz a következménye. (Az előbbi utalások csak a BASIC „lelkivilágának" mélyebb megismerésével válnak világossá. Leírásukat az indokolja, hogy néhány furcsa, a kézikönyvekben nem említett jelenség ezek alapján — némi kísérletezés után — már megérthető.)



8. ábra

A másik módszer szerint a szövegeket a számváltozókhoz hasonló módon tároljuk. Minthogy itt a változók között csak a szöveg hosszát és a memóriabeli címét helyezzük el, így a szöveg hosszának megváltozása nem érinti a változótáblát.

Természetesen ennek a módszernek is van hátránya: az újabb értékadások során a már nem aktuális láncokkal nem törődünk (törlésük túlságosan nehézkes feladat), így előfordul, hogy a sztringterület megtelik. Ilyenkor a már nem aktuális részeket törölni kell, rendezve az összes szövegváltozót. Ha ez alkalmatlan időben következik be, akkor gépünk pl. egy válasz kiírása közben akár néhány másodpercre is „lebénulhat", teljesen megzavarva ezzel a felhasználót.

A fentiek alapján bizonyára könnyen elképzeli az Olvasó a sztringtömbök tárolását is. Ilyenkor — mindkét változatnál — megfelelő számú, a sztringeknél megjelölt adatbyte-csoport tárolódik a tömbben.

Változókból és konstansokból (konkrét számokból vagy szövegekből) kifejezéseket alkothatunk. Ehhez a műveletek végrehajtási sorrendjét kell ismerni. A legtöbb BASIC-értelmező az alábbi sorrendben hajtja végre a különböző műveleteket·

1. Hatványozás.
2. Negatív előjel.
3. Szorzás, osztás.
4. Összeadás, kivonás.
5. Relációjelek.
6. NOT (logikai nem).
7. AND (logikai és).
8. OR (logikai vagy).

Az azonos rangú (prioritású) műveleteket mindig balról jobbra haladva hajtja végre minden BASIC-értelmező. A prioritási sorrendbe olykor a függvényeket is beleveszik, mint a legmagasabb rangú műveletet.

A felsorolásból sejthető, hogy a BASIC-ben a logikai érték is számként ábrá-

zolódik. Ez általában így van, és a hamisnak a Ø számérték felel meg, de az igaz logikai érték ábrázolása már nem egységes.

Természetesen sztringkifejezéseket is alkothatunk, azonban itt csak egyetlen műveletet használhatunk: két szöveg összefűzését a „+" műveleti jellel jelöljük. Az ilyen kifejezésekben sztringfüggvények is szerepelhetnek.

5.2. FELTÉTELVIZSGÁLAT

> *„A számítógép megbízhatatlan, de az ember méginkább."*
>
> *Gilb első megbízhatatlansági törvénye*

Arról már volt szó, hogy a BASIC-program sorokból áll, amelyek növekvő sorszám szerint követik egymást. Egy ilyen utasítássorban több utasítás is lehet: ezeket kettőspontok választják el egymástól.

Kézenfekvő, hogy az utasításokra a soruk sorszámával hivatkozzunk, azonban így — természetesen — nem hivatkozhatunk egy utasítássor nem első helyen álló utasítására. (Minthogy minden utasítás írható külön sorba, így ez nem jelent hátrányt.)

Vezérlésátadáskor (ugráskor) tehát az ugrás után elsőként végrehajtandó utasítás sorszámára hivatkozunk pl. a GOTO kulcsszó után, mindössze arra kell ügyelnünk, hogy ez az utasítás a sor első utasítása legyen.

Ezek után nézzük a BASIC feltételes utasítását! Alakja: IF feltétel THEN 1. utasításcsoport [: ELSE 2. utasításcsoport].

A feltétel teljesülésekor az 1. utasításcsoport hajtódik végre. Itt vezérlésátadás is állhat: sok értelmező megengedi, hogy a THEN kulcsszó után közvetlenül sorszám álljon, azaz ilyenkor a GOTO kulcsszó elhagyható. Az ELSE ág minden esetben elhagyható, sőt néhány rendszer nem is teszi lehetővé a használatát. Ilyenkor, ha a feltétel nem teljesül, akkor a végrehajtás a következő sorral folytatódik.

Az ELSE használata jól szervezett ugrásokkal mindig megkerülhető, de segítségével — ha a rendszer lehetővé teszi — a programot lényegesen egyszerűbbé, áttekinthetőbbé tehetjük: gyakran igen elegáns, tártakarékos kódolást készíthetünk.

A feltétel logikai kifejezés, vagy egész számmá konvertálható (átalakítható) aritmetikai kifejezés lehet. Ez utóbbi esetben a Ø hamisnak, minden más igaznak értékelődik a legtöbb rendszerben.

Néhány BASIC-rendszer az ELSE előtti kettőspontot a belső ábrázolásban maga teszi ki: kiírását nem követeli meg, és ki sem listázza. A feleslegesen kiírt kettőspont azonban az ilyen rendszereket sem zavarja.

Itt érdemes mejegyezni, hogy BASIC-utasítássorba a GOTO utasítás után legfeljebb csak az ELSE ágat érdemes írni, hiszen a sor GOTO utáni része másként sohasem hajtódhat végre.

Feladat. Kódoljuk a 2. ábra algoritmusát!

5.3. A FOR—NEXT-CIKLUS MŰKÖDÉSE; A VEREMTÁR

„Működésképtelen elképzelése mindenkinek van."

Howe törvénye (ismét)

Minden BASIC-rendszer rendelkezik legalább egy ciklusszervező utasításcsoporttal. Ez általában a FOR—NEXT utasításpár, de néhány bővített rendszer egyéb lehetőségeket is biztosít.

A ciklusmagot megelőző utasítást, amely a ciklus szervezését végzi, ciklusfejnek, míg a magot követő, a ciklus végét jelző utsítást cikluszáró utasításnak szokás nevezni.

A FOR—NEXT ciklus alakja:

FOR változónév = kezdőérték TO végérték [STEP-lépésköz]

.

. (ciklusmag)

.

NEXT [változónév]

A ciklusfejben a kezdőérték a végérték és a lépésköz kifejezés is lehet.

A ciklus működése közben egy ún. veremtárat használ. Ez tulajdonképpen a gép RAM-jában kijelölt, speciálisan kezelt tárterület. A verembe adatot csak a többi adatra tehetünk, elérni csak a legfelső adatot lehet; a kiolvasott adat kikerül a veremből. Ezek szerint pl. az utolsó előtti adatot csak úgy olvashatjuk ki, ha előbb az utolsót kiolvassuk (és eldobjuk). (A gyakorlatban ezt úgy oldják meg, hogy egy mutató jelzi a verem mindenkori tetejét.)

Ezek után nézzük a ciklus végrehajtását!

1. A FOR hatására kiértékeli a kezdőértéket, amelyet a változónév által kijelölt változóba, az ún. ciklusváltozóba tölt. Ezután kiértékelődik a végérték, majd a lépésköz, természetesen már a ciklusváltozó kezdőértékének megfelelően. A végérték, a lépésköz értéke, valamint a ciklusmag és ciklusváltozó tárbeli címe elmentődik a veremtárba.

2. Végrehajtja a ciklusmagot.

3. A NEXT-utasítás hatására lezárja az utasításban esetleg szereplő változónévvel bejegyzett, vagy a logikailag utolsó, tehát a verem tetejére bejegyzett ciklust. Azaz növeli a megfelelő ciklusváltozót a hozzá tartozó lépésközzel, megvizsgálja, hogy az új érték túllépte-e a végértéket. Ha nem, akkor a 2. pontra tér, ha igen, akkor törli a ciklus bejegyzését, és a ciklusváltozó értékét megtartva a ciklus utáni utasítással folytatja a végrehajtást.

Figyeljük meg, hogy a FOR—NEXT ciklus hátultesztelő, tehát a ciklusmag egyszer mindenképpen lefut.

A ciklusfejben nem kötelező megadni a lépésközt, ennek az alapfeltételezés szerinti értéke 1.

Néhány BASIC-értelmező nem követeli meg a változónév kiírását a NEXT után. Ilyenkor a legutolsónak bejegyzett ciklus zárásának tekinti az utasítás.

Ha a NEXT utáni ciklusváltozó nem a legutolsónak bejegyzett ciklushoz tartozik, akkor a ciklusváltozónak megfelelő ciklus lezárásával az összes későbbi ciklus bejegyzését törli a rendszer. Természetes az is, hogy nem fordulhat elő két ciklusbejegyzés azonos ciklusváltozóval.

A ciklusba csak a ciklusfejen keresztül célszerű belépni, ellenkező esetben gondoskodni kell arról, hogy a NEXT-utasításra ne kerüljön a vezérlés, különben „FOR nélküli NEXT" hibajelzést kaphatunk.

A (bejegyzett) ciklusból való kiugrást kerüljük, mert ilyenkor a bejegyzés nem törlődik, és ez később bonyodalmakat okozhat.

Itt jegyezzük meg, hogy néhány BASIC-rendszer a leírtaktól némileg eltérő működésű FOR—NEXT ciklust használ, ezeket azonban nem tárgyaljuk.

Feladat. Kódoljuk a 3*c* ábra algoritmusát!

Feladat. Írjuk meg a rendező modulokat! (Az A(I) és az A(J) változók felcserélését az A = A(I):A(I) = A(J):A(J) = A utasítássorozattal végezhetjük.)

5.4. A DO—LOOP-CIKLUS LEHETŐSÉGEI

> *„Mindent lehet egyszerűbben is csinálni."*
>
> *Iles törvénye (ismét)*

A bővített BASIC-rendszerek jelentős része tartalmaz további ciklusszervező utasításokat. Ezek megvalósítása rendkívül sokféle, működésük lényege azonban eléggé egységes. Ezért csak a C16-os számítógépen is meglévő megoldást tárgyaljuk, hiszen az egyéb megoldások ettől tulajdonképpen csak a formalizmusban térnek el.

A DO—LOOP-utasításpárral tetszőleges helyen (akár több helyen is) tesztelő, vagy végtelen ciklust szervezhetünk.

A fejben való tesztelés a DO utáni, a ciklus végén való tesztelés a LOOP utáni feltételmegadással írható elő. Mindkét feltétel lehet UNTIL- vagy WHILE-típusú. Az első esetben a ciklus addig folytatódik, míg a kulcsszó utáni feltétel igazzá nem válik (a kifejezés értéke nullától különböző lesz), míg a második esetben a ciklus addig folytatódik, amíg a feltétel igaz (nullától különböző).

A ciklusmagban az EXIT-utasítás segítségével szervezhetünk tesztelést. Ha a vezérlés erre az utasításra kerül, akkor a ciklus végrehajtása befejeződik, a ciklus bejegyzése törlődik. Az EXIT-utasítás nyilvánvalóan valamilyen feltételvizsgálat megfelelő ágába írandó.

A DO—LOOP-ciklus működése vázlatosan a következő.

1. A DO hatására bejegyzi a ciklust: a verembe teszi a ciklusfej címét és az aktuális sorszámot.

2. Megvizsgálja a DO utáni feltételt: van-e, és ha van, mi a teendő. Ennek megfelelően elkezdi a mag végrehajtását, vagy ha be kell fejezni a ciklust, megkeresi a programszövegben a következő LOOP kulcsszót, törli a ciklus bejegyzését és a következő utasításra (nem utasássorra!) tér át.

3. Ha EXIT-utasításra fut a magban, akkor megkeresi a programszövegben a következő LOOP kulcsszót, törli a ciklusbejegyzést és a következő utasítással folytatja a végrehajtást.

4. A LOOP hatására megvizsgálja a feltételt: ha nincs, vagy a ciklust folytatni kell, akkor a veremből kiolvasott címre való ugrással a 2. pontra tér, különben törli a ciklusbejegyzést és a következő utasításra tér.

A DO—LOOP-ciklusok természetesen egymásba ágyazhatók.

Az elmondottakból látható, hogy amíg a FOR—NEXT-ciklus több helyen (több NEXT-utasítással) is lezárható, addig ez a DO—LOOP-ciklus esetén általában nem megy. Másrészt az is nyilvánvaló, hogy a ciklusba való beugrások és a magból való kiugrások igen veszélyesek, mert az összetartozó DO- és LOOP-utasítások könnyen összekeverednek.

5.5. SZUBRUTINOK, SZUBRUTINSZERKEZETEK

> *„Aki utoljára nevet, valószínűleg nem értette meg a viccet."*
>
> *Bocklage törvénye*

A BASIC lehetővé teszi alprogramok, eljárások szervezését. Erre speciális utasításpár szolgál, amelynek működése a BASIC-ben szokatlan módon egységes. (Itt jegyezzük meg, hogy gépi kódú alprogramok is használhatók a BASIC-ből, erre azonban nem térünk ki, mert ehhez sokkal mélyebb ismeretekre lenne szükség.)

Szubrutin (alprogram, eljárás) végrehajtását „GOSUB sorszám" alakú utasítással kezdeményezhetjük, míg a rutin végét a RETURN-utasítás jelzi. A legtöbb rendszer megköveteli, hogy a hivatkozott sorszám létezzék, más rendszerek ilyenkor a következő létező sorszámra értelmezik a hivatkozást. Ez utóbbiak gyakran azt is megengedik, hogy a sorszámot aritmetikai kifejezés írja elő. A RETURN nem tartalmazhat semmilyen paramétert.

A szubrutinra való ugráskor a gép bejegyzi a hívást: a verembe teszi a visszatérés (tárbeli) címét és az aktuális sorszámot, majd az utasításban kijelölt sorszámú sorra ugrik.

A szubrutint lezáró RETURN-utasítás hatására visszatöltődik a tárolt aktuális sorszám és a végrehajtás a bejegyzett tárcímen folytatódik, a bejegyzés pedig törlődik. A visszatérés nyilvánvalóan mindig az utolsó érvényes bejegyzésnek megfelelően hajtódik végre.

A szubrutin a (hozzá tartozó) RETURN-utasításig tart, kiugrani *nem lehet* belőle: az ugrás utáni programrész is a szubrutin részének értelmeződik. (Megjegyezzük, hogy néhány BASIC-értelmező lehetővé teszi a veremtár ürítését, ami a szubrutinüzemmódot megszünteti, így tehát valójában ilyenkor sem a szubrutinból ugrunk ki, hanem magát a szubrutint szüntetjük meg.)

Egy szubrutin akárhányszor meghívható, sőt szubrutinból (további) szubrutinok is hívhatók, azaz a szubrutinok egymásba ágyazhatók. Ilyenkor a visszatérés mindig az utolsó hívás helyével kezdődik.

A szubrutinok használatára vonatkozó példák a később közölt programokban találhatók.

A figyelmes Olvasónak bizonyára feltűnt, hogy a ciklusok és a szubrutinok szervezése a számítógép veremtárának használatával történik. Ez biztosítja, hogy akár a ciklusok, akár a szubrutinok (szinte korlátlanul) egymásba ágyazhatók, méghozzá vegyesen is. Éppen ez utóbbi lehetőség vet fel egy olyan problémát, ami — bár ritkán fordul elő — a BASIC-programozásban és kódolásban kevéssé járatosaknak szinte megoldhatatlan nehézséget jelent. (Ilyen jellegű hibákat szoktak elkövetni azok a

már ügyesebb számítógépkedvelők is, akik a számítógép mellett kezdenek a program írásához.)

Jegyezzük meg, hogy *minden ciklusban, ill. szubrutinban a megnyitott ciklus, ill. szubrutin teljes egészében benne van.*

Ez azt jelenti, hogy hibajelzést kapunk, ha egy megnyitott ciklusban szubrutint hívunk és ebben zárjuk le a ciklust, vagy ha a bejegyzett szubrutinban megnyitunk egy ciklust, amelyet a visszatérés után zárunk csak le. Hasonló a helyzet ciklusok hibás egymásba ágyazása esetén; ez ellen például a FOR—NEXT-ciklus a már leírt módon védekezik, vagy hibát (is) jelez. Szubrutinokat pedig nem is lehet hibásan egymásba ágyazni.

A fenti „programszerkezetek" hibája nyilvánvalóvá válik, ha megpróbáljuk a folyamatábrájukat elkészíteni. Így az ellentmondásosságuk könnyebben felismerhető.

Egyébként éppen a felsoroltak a BASIC struktúraszervező utasításcsoportjai.

5.6. A PROGRAM ADATAI: DATA—READ—RESTORE

> *„Amikor elraktározol valamit a gép memóriájában, raktározd el saját memóriádban, hogy hova tetted."*
>
> *Leo Beiser első számítógép-axiómája*

A BASIC lehetővé teszi a futás során használandó adatok tárolását a programban. Ezek a programszövegben helyezkednek el és akkor olvashatjuk ki, ha szükségünk van az ilyen adatra.

Az adatokat DATA-utasításokban adjuk meg: a kulcsszó után vesszővel elválasztva soroljuk fel — megfelelő sorrendben — az adatokat. Ezek szöveges vagy számtípusú adatok lehetnek. Minthogy az utasítást egyéb utasítások követhetik a sorban, így a vesszőt, ill. kettőspontot tartalmazó szövegeket idézőjelek közé kell tenni. Végrehajtáskor a gép a DATA-utasításokat átlépi.

Az eddigiekből adódik, hogy DATA-utasítást a programban bárhol elhelyezhetünk, de ezt is lehet nagyon ügyetlenül tenni. Általában érdemes az adatokat a program végére írni, csak nagyon ritkán célszerű a program belsejében elhelyezni. (Erről később lesz még szó.)

Az adatok olvasására a READ-utasítás szolgál. Egy READ után több változót is felsorolhatunk, ezekbe fognak töltődni a DATA-utasítások soron következő adatai. Újabb READ-utasítás hatására az olvasás tovább folytatódik.

A program írásakor ügyelni kell arra, hogy a felsorolásnak megfelelően olvassuk az adatokat, ha ugyanis a READ-utasításban megadott változó(k) típusához nem illeszthető a soron következő adat, vagy az adatok elfogytak, akkor hibajelzést kapunk.

A READ-utasítás működése a következő: az első olvasás előtt egy, a rendszer által kezelt mutató (pointer), az ún. DATA-mutató a program elejére mutat. Az első olvasáskor az értelmező elindul a DATA-mutató által meghatározott címről és megkeresi az első DATA-utasítást. Ezután beolvassa az első adatot és a mutatót az adat utáni byte-ra állítja. Minden további olvasáskor két eset lehetséges: a mutató egy adatlistában levő vesszőre mutat, vagy pedig utasításvégen (kettőspontok vagy sorvégen) áll a DATA-mutató. Az első esetben a beolvasás a következő byte-on foly-

tatódik, míg a második esetben az értelmező ismét DATA-utasítást keres, és az azt követő adatot olvassa. Ha keresés közben eléri a programvéget, akkor „nincs több adat" hibajelzést ad az olvasás sorára.

Ezek után érhető a RESTORE-utasítás működése. Hatására a DATA-mutató a programszöveg elejére (egészen pontosan az első sor előtt tárolt sorvége jelre) áll. Ha az értelmező megengedi a RESTORE után sorszám megadását is, akkor ilyen esetben a végrehajtáskor a gép megkeresi a megadott sorszámú sort és a mutatót a sort megelőző byte-ra, azaz az előző sor sorvégjelére állítja.

5.7. ADATBEVITEL: GET, INKEY$, INPUT

> *„Ha az inputellenőrzést úgy tervezték meg, hogy kiszűrje a téves inputokat, egy lelemények idióta biztosan kidolgozza azt a módszert, melynek segítségével a téves inputok megkerülik az ellenőrzést."*
>
> *Troutman ötödik programozási posztulátuma*

A BASIC-értelmezők általában kétféle lehetőséget biztosítanak programfutás közbeni adatbevitelre. Ezek egyike az INPUT-utasítás, amely (több karakterből álló) sor bevitelére szolgál: a végrehajtáskor a gép felfüggeszti a program futását és az adat bevitelére vár. A futás csak az adatbevitel után folytatódik: a bevitel végét sorzárás jelzi. Egy INPUT-utasítással több adatot is bevihetünk, ilyenkor az adatokat vesszővel elválasztva kell az utasítás hatására kiírt kérdőjel után felsorolni. A megadott adatokat az INPUT után felsorolt változókba tölti a gép, ilyenkor a tétellistának megfelelő számú és típusú adatot vár.

Ha számváltozóba való beolvasáskor olyan karaktersorozatot adunk meg válaszként, amely a szóközöktől, az előjelektől és a jegyek számától eltekintve megfelel a gép nyomtatási formátumának, akkor ez konvertálódik (átalakítódik a belső ábrázolásnak megfelelő formára), esetleg túlcsordulási hiba keletkezik. Néhány értelmező aritmetikai kifejezést is elfogad ilyen esetben. Ha a formátum nem ilyen, akkor a legtöbb rendszer (hibaüzenet után) az adatbevitel megismétlésére vár. Az ilyen hiba megelőzhető, ha a számot szövegváltozóba visszük és utána alakítjuk át (a később tárgyalandó VAL függvény segítségével) számmá, és így töltjük a számváltozóba.

Ha szövegváltozóba vesszőt tartalmazó karaktersorozatot akarunk tölteni, akkor azt idézőjelek közé kell tenni, ellenkező esetben több adatként értelmeződik. Hasonló a helyzet, ha szám bevitelekor a tizedespont helyett vesszőt írunk. Ilyenkor általában jelzi a gép az adathibát, de a program tovább fut. Ha a rendszer szövegváltozónál kiírja az idézőjelet, akkor a fenti hiba nem fordulhat elő, ilyenkor ugyanis minden adat bevitelét a sorzárással kell jelezni, és ez értelemszerűen a számváltozókra is vonatkozik.

Néhány rendszer beépített védelemmel rendelkezik, ami azt jelenti, hogy formai hibás adatot nem fogad el: többnyire a hiba vélt helyét is megadja, megjelölve az adatnak azt a részét, amit már nem tudott értelmezni.

Az INPUT-utasításhoz nagyon hasonlóan működik a GET-utasítás, ez azonban egyetlen karaktert, ill. jelet visz be. Végrehajtásakor a program futása nem áll le,

hanem az értelmező egy puffer tartalmát vizsgálja. Ha a puffer üres, akkor a GET után álló szövegváltozó tartalma az üres szöveg, a számváltozóé pedig nulla lesz.

Általában a GET-utasításban is szerepelhet változólista. Ilyenkor a billentyűzet-puffer tartalmát sorra beolvassa a változókba.

Ha számváltozóba olyan karaktert olvastatunk be, amely szám nyomtatási képében nem fordulhat elő, akkor általában valamilyen (többnyire típuskeveredés) hibazelzést kapunk.

Szövegváltozóba való bevitelnél a vessző és a kettőspont, tehát a listák határolójelei okozhatnak hibát, bár ezt a legtöbb rendszer helyesen kezeli.

Az INPUT- és a GET-utasítások használhatók külső eszközökről (magnetofon, lemezegység) történő adatbevitelre is, ilyen esetben a kulcsszót egy kettőskereszt és az eszközkijelölés követi, majd ezután a változólista áll.

Végül a harmadik adatbeviteli lehetőség egy, a GET-utasításhoz hasonlítható függvény, az INKEY$. Minthogy ez argumentum nélküli függvény, így formalizmusa alapvetően eltér az előző két utasításétól. Az értékadás bal oldalán álló változóba betölti az utoljára lenyomott billentyű által meghatározott egykarakteres szimbólumot. A változók és a bevitt adat típusára vonatkozóan a GET-utasításnál leírtak adnak eligazítást.

Az INKEY$ függvény általában egyetlen byte-os tárolómezőt használ, amely függvényhivatkozáskor kiürül, azaz tartalma nulla lesz, és ez csak a következő billentyű lenyomásával változik meg, amikor is a lenyomott jel kódja kerül a tárolómezőbe. Természetesen minden újabb billentyű lenyomásakor a tárolómező tartalma felülíródik, de nulla csak a kiolvasás (függvényhivatkozás) után lesz ismét.

A számítógépek az egykarakteres bevitelre a GET-utasítás és az INKEY$ függvény valamelyikét tartalmazzák.

5.8. NUMERIKUS FÜGGVÉNYEK, FELHASZNÁLÓI FÜGGVÉNYEK

> *„Bármely program bonyolultsága addig fokozódik, amíg túl nem nő programozója képességein."*
>
> *A számítógépes programozás 7. törvénye*

A függvényeket alapvetően kétféle szempont szerint csoportosíthatjuk: a függvény által szolgáltatott érték, valamint a függvény argumentuma(i) alapján.

Az első szempont szerint egy függvény vagy numerikus, vagy szöveg-, vagy speciális függvény lehet.

A másik szempont szerint numerikus értékeken (számokon), szövegeken, vagy több, ill. többféle értéken értelmezett függvényekről szokás beszélni. Az első két csoportba tartozó függvények egy argumentumtól függenek (egyváltozós függvények), míg a harmadik csoportba tartozók többtől (többváltozós függvények).

A numerikus értékeken értelmezett függvények közé tartoznak az ún. matematikai függvények: hatványfüggvény, logaritmusfüggvény, szögfüggvények, abszolútérték-függvény, előjelfüggvény, négyzetgyökfüggvény, az egészrész és a szögfüggvények inverzei közül legalább az arkusztangensfüggvény. Ezek értelmezése a mate-

matikában megszokottnak felel meg, bár néhány sajátosság előfordulhat. A hatványozás és a logaritmus mindig természetes alapú, a szögfüggvények ívmértékkel dolgoznak. Előfordulhat, hogy a szögfüggvény csak bizonyos tartományba eső szögeket fogad el, vagy csak pozitív szám arkusztangensét számolja ki az értelmező.

Ide tartozik néhány speciális függvény is, így pl. a PEEK (cím) függvény, amely a cím által kijelölt byte tartalmát adja, de a véletlenszám-függvény és az állapotot megadó függvények is.

Végül itt kell szólni a típusátalakítást végző függvényekről (konverziós függvények), melyeket általában csak professzionális gépekbe építenek be. Ezek a különböző számváltozók közötti típusátalakítást végzik. Vigyázzunk, mert az egészrész-függvény nem konverziós függvény, ugyanis a függvényérték továbbra is valós (vagy megfelelő) típusú lesz, csak az értéke lesz egész szám, de egész típusú számként (16 biten) esetleg nem is ábrázolható!

Szövegeken értelmezett numerikus függvény a VAL függvény, amely egy számszöveg számértékét adja meg. Ez tulajdonképpen egy speciális konverziós függvény. Ilyen még az ASC és a LEN függvény; ezek az argumentumszöveg jellemzőit adják értékül. Az ASC függvény a szöveg első karakterének (ASCII) kódját, a LEN függvény pedig a szöveg hosszát szolgáltatja. (Azok a gépek, amelyek a belső ábrázolásban nem az ASCII kódokat használják, az ASC függvény helyett más függvénnyel — pl. CODE — állítják elő a karakterkódokat.)

A BASIC rendszerek általában csak egy többváltozós numerikus függvényt használnak. Ez a második szövegnek az elsőben való első előfordulásának kezdőpozícióját adja. Gyakran megadható még egy harmadik argumentum is, amely szám, és a keresés kezdetének a helyét jelöli ki. Ez a függvény az INSRT vagy a POS.

A legtöbb BASIC lehetővé teszi felhsználói függvények bevezetését. Ezek többnyire csak numerikusak lehetnek, és a változók számára, ill. típusára is vonatkozhatnak kötöttségek. A már definiált függvényekre a továbbiakban ugyanúgy hivatkozhatunk, mint a beépített függvényekre.

Végül megjegyezzük, hogy néhány függvény ún. álargumentumot használ, azaz az argumentum értékére nincs szükség. Azonban az ilyen függvények is ellenőrzik az argumentum típusát (ezt olykor fel is használják). Vannak továbbá argumentum nélküli függvények is, ilyen pl. a legtöbb gépen a PI, amelyet a felhasználó ugyan beépített konstansnak érez, de a belső kezelése a függvényekével egyezik meg.

5.9. SZÖVEGFÜGGVÉNYEK

> *„Soha nincs rá idő, hogy jól csináljuk, de arra mindig van idő, hogy újracsináljuk."*
>
> *Meskimen törvénye*

Szövegek kezeléséhez — a már említett néhány numerikus függvényen kívül – szöveget (karakterláncot) szolgáltató függvényt is használhatunk.

Ezek között néhány speciális, inkább csak a bővített rendszerekben használt függvénytől eltekintve (pl. a HEX$) csak kettő egyváltozós függvény használatos. Egyik az argumentumban megadott kódú karakterből álló, egy hosszúságú szöveget szolgáltat. Neve a BASIC-rendszerekben általában CHR$; még akkor is ezt használják, ha a gép belső ábrázolása eltér az ASCII kódtól.

A másik az argumentumként megadott aritmetikai kifejezés számértékének nyomtatási képét adja karakteres formában. Jele: STR$.

Szöveg argumentumú egyváltozós szövegfüggvény nem szükséges, ugyanis ezek bármelyike valamelyik többváltozós függvény speciális eseteként kezelhető.

A többváltozós függvények egyik csoportja az első argumentum egy szeletét adja; balról, jobbról vagy középről. A másik csoport függvényei valamilyen speciális karaktersorozatot állítanak elő: ismétlődő karakterből vagy karaktercsoportból álló láncot, ill. egyéb gépspecifikus karakterláncokat (pl. ilyen az ABC 8Ø nagypontosságú aritmetikája).

5.10. SPECIÁLIS FÜGGVÉNYEK

„Tévedni emberi dolog, de igazán összekutyulni valamit csak számítógéppel lehet."
A megbízhatóság ötödik törvénye

Ide elsősorban azokat a függvényeket soroljuk, amelyek a gép állapotát változtatják meg. Ilyen pl. a TAB függvény, ez a kurzort az aktuális sor argumentumban megadott pozíciójára viszi (ha azt még nem lépte túl).

A több gépen is megvalósított SPC függvény a kurzor relatív helyét változtatja meg: az argumentumban meghatározott számú jobbra léptetést végez. (Itt hívjuk fel a figyelmet arra, hogy az SPC$ függvény szövegfüggvény, amely az argumentumban megadott számú szóközből álló karakterláncot adja. Ezt a függvényt azonban csak kevés BASIC-rendszer ismeri.)

Csaknem minden rendszer tartalmaz egy gépi kódú alprogram aktivizálását végző függvényt. Ez általában egyetlen argumentumot fogad, és a BASIC és a gépi kódú rutin közötti adatátvitelt szolgálja. A fordított irányú átvitelre a függvény értéke használható. Ez természetesen már nem tartozik a BASIC programozás tárgykörébe.

A speciális függvények közé szokás sorolni az argumentum nélküli függvényeket. Ezek az argumentum formalizmusát sem kívánják, így a felhasználó általában nem veszi észre, hogy függvénnyel dolgozik. Ennek ellenére jó tudni, hogy a gép ezeket a függvényekkel azonos módon kezeli, pl. a prioritási kérdések miatt.

Ilyen függvény a már korábban említett INKEY$ és a legtöbb gépen a PI vagy π.

5.11. HIBAKEZELÉS A PROGRAM FUTÁSA KÖZBEN

„Sohase tárj fel olyan hibát, amellyel nem tudsz mit kezdeni!"
Steinbach rendszerprogramozási útmutatása

Szinte minden BASIC-rendszer lehetővé teszi a program futása közben keletkezett hibák kezelését. Ennek az a célja, hogy a külvilággal lebonyolított adatforgalom során fellépő különböző hibaállapotok ne a programfutás megszakítását eredményezzék, hanem egy olyan rutin aktivizálódjék, amely alkalmas a hiba okának felderítésére és a hibaállapot megszüntetésére.

A hibakezelés tehát nem a programozó hibái ellen nyújt védelmet, éppen ellenkezőleg. A programozónak kell olyan hibakezelő rutint írnia, amely a programban előforduló hibáknak megfelelő módon elvégzi a szükséges adat-, ill. állapotmódosításokat.

Hibakezelést célszerű alkalmazni adatvédelem céljára akkor is, ha sok adatot kell feldolgozni, és az adatok egyenkénti ellenőrzése nagyon időigényes lenne. Ilyenkor érdemes elkezdeni a még ellenőrizetlen adatok feldolgozását, és csak akkor kezdeni a hibavizsgálatot, amikor a hiba már bekövetkezett.

Könnyen belátható, hogy ismeretlen szerkezetű adatállományok feldolgozásánál sokszor ez az egyetlen lehetőség a feladat megoldására: az utolsó adat utáni sikertelen olvasási kísérlet hibát okoz, és ezt felismerve tudhatja a program, hogy a feldolgozandó adatok végére ért. (Az ilyen adatállomány-kezelések azonban csak haladóbb szinten és megfelelően kiépített rendszereken oldhatók meg.

Ehhez az Irodalomjegyzékben felsorolt könyvekben talál részletesebb leírásokat az Olvasó.)

6. A PROGRAM ÉS AZ ADATOK TÁROLÁSA

„Jobb ma egy jó terv, mint holnap egy tökéletes."

Patton törvénye

Bár a BASIC szintű programozáshoz nem szükséges a tárolási elvek ismerete, mégis hasznosnak látszik ezeknek a vázlatos leírása.

A mikroszámítógépek BASIC-rendszereit csak úgy tudjuk igazán jól kihasználni, ha a működés hátterét is értjük. Ez a BASIC-trükkökhöz elengedhetetlen. Azt is meg kell azonban jegyezni, hogy ezen leírásokban a C16-os gép szolgált alapul, így csak a hasonló rendszerekről lesz szó.

6.1. A PROGRAM A TÁRBAN

„Sose törd a fejed olyasmin, amit biztosan lehet tudni!"

De Never elkallódott törvénye (ismét)

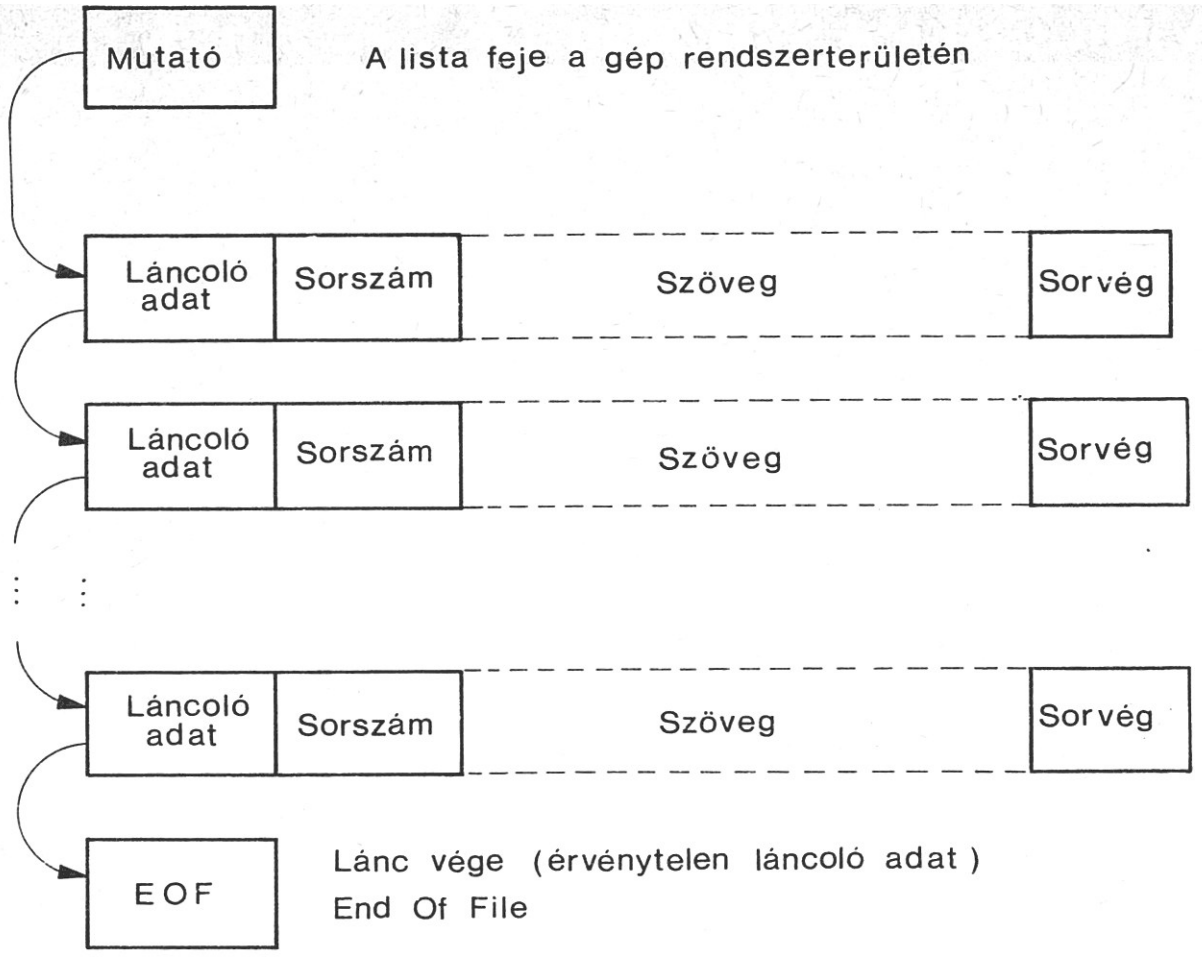
Tudjuk, hogy egy BASIC-program egymást követő utasítássorokból áll, az utasítássorokat pedig — szintén egymást követő — utasítások alkotják. A sorokat a sorszámukkal azonosíthatjuk. Mindezek alapján tehát egy programot kézenfekvő a növekvő sorszámok szerinti sorrendben tárolni, különösen azért, mert ez a végrehajtás sorrendje is.

Az is érthető, hogy a sorokat el kell határolni egymástól, azaz ún. sorvégjelet érdemes használni, hiszen az utolsó sort nem zárhatnánk le semmivel, ha a sorok elejét jelölnénk meg.

Az eddigiek szerint tehát a program a tárolás szempontjából olyan rendezett adathalmaznak tekinthető, amelynek adatelemei az utasítássorok. Ezek mindegyikét adatvégjel zárja. Az adatok soros tárolása miatt tulajdonképpen semmi más nem kell ahhoz, hogy egy programot tárolni tudjunk. Az elmondottnál bonyolultabb tárolást csak azért használnak a BASIC-értelmezők, mert a keresés egy ilyen adathalmazban igen nehézkes lenne.

Minden sort ellátnak egy fejrésszel. Ebben tárolják az utasítássor sorszámán kívül a sor hosszát jellemző adatot is. Ezt alapvetően kétféleképpen oldják meg. A gyakrabban használt megoldásban a sor fejrésze tartalmazza a következő sor tárbeli címét (pointert), míg a másikban a sor által elfoglalt byte-ok számát tárolják a fejrészben. Ezzel egy *láncolt adatábrázolást* hoznak létre (9. ábra).

A program vége arról ismerhető fel, hogy az utolsó sor után egy érvénytelen mutató, vagy nulla sorhosszmegadás áll. Ezeket „lánc vége" jeleknek nevezzük (EOF — *E*nd *O*f *F*ile).



9. ábra

A fentiek figyelembevételével tehát egy vezérlésátadás a következőképpen történik.

Összehasonlítja a hivatkozott és az aktuális sorszámot. Ha a hivatkozott a nagyobb, akkor megkeresi a sorvéget, és a következő sorra lépve elkezdi a sorszám keresését. Ha a hivatkozott sorszám nem nagyobb, akkor elmegy a program elejére (ezt egy, a rendszer által kezelt mutató jelöli ki, és ez a lánc vagy lista feje), és innen kezd keresni. Megnézi a sor sorszámát, hogy a keresett szám előtt van-e még. Ha igen, akkor a mutató vagy a sorhossz segítségével átlépi a sort, és folytatja a keresést. Ha már nem előtte van, akkor megvizsgálja, hogy túllépte-e. Ha igen, akkor nincs ilyen számú sor, ha nem, akkor megtalálta; a végrehajtás innen folytatódik.

Vegyük észre, hogy a sor keresésének időtartama erősen függ attól, hogy hány sort kell átlépni a kereséskor. Éppen ezért érdemes kevés sorba írni a programot. Másrészt az is nyilvánvaló, hogy több utasítást egy sorba írva a tárral is takarékoskodunk, hiszen minden sor szervezéséhez két byte kell a sorszám tárolására, egy vagy kettő a sorhossz vagy a mutató számára, végül egy byte a sorvégjel. Ha már megnyitott sorba írunk újabb utasítást, akkor — az utasítás byte-jain kívül — egyetlen kettőspontot kel tárolni az előző utasítástól való elválasztásra. Ez tehát legalább három vagy (mint a C16 esetében) négy byte megtakarítását jelenti.

6.2. A VÁLTOZÓK HELYE A TÁRBAN

„Ha valamit érdemes véghezvinni, akkor túlzásba vinni is érdemes."
Leo Rogers áldása a második részre

Az eddigiek alapján nyilvánvaló, hogy a változók tárolása esetén is a programtárolásnál megismert elveket kell alkalmazni, hiszen a programok rendkívül sokszor keresnek változót, ezt a keresést tehát feltétlenül segíteni kell.

A BASIC-rendszerek a változókat általában két listában tárolják. Az egyik az egyszerű változók, a másik a tömbváltozók táblázata (ebben a sorrendben).

Az egyszerű változók láncolását kétféle módszerrel végezhetik. Mivel az egyszerű változó hosszát egyértelműen meghatározza a típusa, így ez használható a következő adat címének előállítására. A második módszer egyszerűbb, gyorsabb, de a tárterületet pazarolja. Ekkor ugyanis az egyszerű változókat típusuktól független nagyságú területen tárolják. Ilyenkor a változó típusának megfelelő hosszmegadást elhagyva nem nő meg a keresési idő, ami tehát változónként egy-egy byte megtakarítását eredményezi. Figyeljük meg itt is, hogy a megoldások valamilyen kompromisszum eredményeként jönnek létre.) Könnyű belátni, hogy a második módszert olyan rendszerek esetén célszerű alkalmazni, amelyek közel azonos hosszúságú adatokat használnak, míg az elsőt akkor, ha pl. dupla pontosságú változókat is kezelni kell (mondjuk egy professzionális rendszerben), vagy akkor, ha a szövegváltozók karakterei a változótáblázatban tárolódnak a *8a* ábra szerint.

A tömbváltozók tárolása egyértelműbb, hiszen ezek mérete változó, így a tömb jellemzői között feltétlenül szerepelnie kell egy, a méretre vonatkozó adatnak; a láncolást ez végzi.

Minthogy a változókkal kapcsolatban az egyik művelet a bejegyzés, ezért érdemes a változótáblázat végének a címét tárolni. Ez a gyakorlatban úgy történik, hogy az egyszerű változók táblázatának kezdőcímét, a tömbváltozók táblázatának a kezdőcímét és a végcímét tárolják. A megfelelő kezdőcímek lesznek a listák fejei, míg a tömbváltozó-táblázat adatai helyettesítik a „lánc vége" jeleket. Így az újabb változó bejegyzésekor nem kell keresni a táblázat végét, ahová a változó elhelyezhető, viszont keresésekor — amikor a változó tárbeli címét kell előállítani — a listák ilyen megadása semmilyen hátrányt nem okoz.

Megjegyezzük, hogy a program esetén alapvetően más a helyzet, ugyanis a program végét mindig szekvenciálisan (sorban egymás után) végrehajtott művelet során (listázás, futás, sorszámkeresés) érjük el. Ekkor címvizsgálat nélkül kell észrevenni, hogy a lista végére értünk, ellenkező esetben minden sor végrehajtása után ellenőrizni kellene, hogy nem értük-e el a program végét.

Ezek után érthető az a BASIC-sajátosság, hogy ciklusváltozóként csak egyszerű változó használható. Ennek oka az, hogy a program futása során az egyszerű változók helye állandó, hiszen ekkor már sem a program, sem a táblázatban előtte álló változók mérete nem változik.

Ugyanekkor az is nyilvánvaló, hogy a tömbváltozók újabb egyszerű változó bevezetésekor elmozdulnak a tárban. A ciklusváltozó veremben tárolt címe tehát nem azonosíthat tömbelemet.

6.3. A PROGRAMJAVÍTÁS KÖVETKEZMÉNYEI

„Előbb-utóbb biztosan fellép a körülmények lehető legrosszabb elrendeződése."
Pech második törvénye

Most már bizonyára az Olvasó is fel tudná sorolni az említett következményeket.

Nyilvánvaló, hogy a program javításakor megváltozhat a ciklus címe, a megnyitott szubrutin visszatérési címe és a ciklusváltozó címe, így tehát feltétlenül törölni kell a vermet.

Előfordulhat az is, hogy a javítás során megszüntetünk egy-egy változóra vonatkozó minden hivatkozást, így tehát a változótáblázatokat is érdemes törölni.

Már az első ok is elegendő ahhoz, hogy javítás után egy programot ne lehessen folytatni, ekkor viszont a változók bejegyzése tulajdonképpen felesleges, ezért törli a változótáblákat is a legtöbb BASIC-rendszer.

6.4. A VÁLTOZÓK KEZELÉSE

„Nehéz a sasokkal szárnyalni, ha pulykákkal vagy körülvéve."
Langsam ornitológiai axiómája

A változókkal végzett műveletek közül kettőt már megismertünk. A harmadikat ritkábban emlegetik, pedig ez is érdekes. Felvetődik ugyanis a kérdés, hogy miért nem engedi meg a BASIC egy vagy néhány változó törlését. Az ok igen egyszerű, ugyanis ekkor megváltozhatna a ciklusváltozó helye, de akár törölhetnénk is az aktuális ciklusváltozó bejegyzését.

Törölni tehát csak az összes változót lehet, ekkor azonban törölni kell a ciklusvermet, emiatt viszont a visszatérési vermet is. Így tehát ciklusban, ill. szubrutinban nem szabad törölni a változókat, ez ugyanis — a verem törlése miatt — (lezáráskor) hibát okoz.

Megjegyezzük, hogy a tömbváltozók törlése (akár egyenként is) elvileg lehetséges lenne, a BASIC mégis igen szigorú kötöttségeket ír elő a tömbök kezelésére, ami — minden bizonnyal — nagygépes örökség.

Jó tudni, hogy a változók törlése csak logikai; az történik mindössze, hogy a tömbváltozók kezdetét és végét jelző mutatókat az egyszerű változók kezdetének a címére állítja.

7. A C16 KULCSSZAVAI

„Kérdéseidre nincsenek válaszok, csak keresztutalások."

Weiner könyvtári törvénye

A kulcsszavak leírásának célja az alapgép programozásának segítése. Ebből adódik, hogy a BASIC kulcsszavak szabatos, pontos leírása az elsődleges.

Minden kulcsszó teljes alakja mellett megtalálható a COMMODORE által használt rövidítés is (ha van), továbbá a kulcsszó jellege, és valamilyen magyarázat, fordítás. A kulcsszót jellemző betűcsoportok jelentése:

F	különleges függvény;
K	utasításban, parancsban használt kulcsszó;
M	művelet (operátor);
NF	számokon értelmezett numerikus függvény;
NSF	számokon értelmezett szövegfüggvény;
P	parancs;
SNF	szövegeken értelmezett numerikus függvény;
TF	többféle változón értelmezett függvény;
U	utasítás.

A leírásban nem szerepelnek a matematikában is használatos *műveleti és relációjelek*. Ezekről korábban már volt szó, így itt csak azt jegyezzük meg róluk, hogy mindegyiküket műveletet előíró kulcsszóként kezeli a gép, de az egyenlőségjelnek egy további funkciója is van, ez az értékadás. Pl. a 10 A = A = 0 értékadó utasítás formailag (szintaktikusan) helyes; az első egyenlőségjel értékadást ír elő, a második relációjelként értékelődik. Az utasítás után az *A* változó értéke −1 lesz, ha előzőleg 0 volt, és 0 lesz minden egyéb esetben. A relációjeleket összehasonlítási műveletként foghatjuk fel, amelyek eredménye igaz (−1) vagy hamis (0).

A *BASIC függvényeinek* argumentumait zárójelbe kell tenni. A nyitó zárójel közvetlenül a függvény neve után írandó, különben formai hibát jelez a gép. A függvények értékét természetesen csak a megfelelő típusú változóba szabad tölteni, ellenkező esetben típuskeverési hiba keletkezik. Az argumentumok általában kifejezések is lehetnek, de típusaik és sorrendjük kötött.

Az *álargumentumot* használó függvények formai elemzésekor a megfelelő típusú argumentumot várja az értelmező, ezek megadása tehát kötelező.

Néhány utasításban bizonyos részek elhagyhatók. Ilyenkor a rendszer a hiányzó részeket az *alapfeltételezés* szerinti értékekkel helyettesíti, amelyet kissé pongyolán (de röviden) lapértelmezésként fogunk emlegetni.

Azokat a karaktereket, melyeket a számok leírására használunk, célszerű külön névvel ellátni. A gép ugyanis *számkarakterekként* kezeli a számjegyeket, az előjeleket,

az E-betűt és a (tizedes)pontot. Ezek kezelése adatbevitel esetén a többi karaktertől eltérően történik.

A C16 sor beírásakor csak 64 000-nél kisebb *sorszámot* fogad el, de átsorszámozáskor ennél nagyobb sorszám is előállhat. Sorszámra való hivatkozásban általában csak létező sorszám állhat, ellenkező esetben hibajelzést kapunk.

A szövegkonstansokat általában idézőjelek között kell megadni. Adatbevitel esetén és a DATA-utasításokban az idézőjel néhány kivételtől eltekintve szöveg megadásakor sem szükséges.

A C16 BASIC értelmezője minden esetben egészrészt vesz, ha az adott helyen csak egész szám állhat, erről tehát nem kell gondoskodnunk.

A C16 igen sokoldalúan használható grafikus *utasításkészlettel* rendelkezik. Ez lehetővé teszi, hogy egyszerű BASIC programrészletekkel is bonyolult grafikus ábrákat rajzoltathatunk, ráadásul elég gyorsan is. Minderről azért érdemes külön szólni, mert ezek a grafikus utasítások a különböző könyvekben leírtaknál sokkal többet tudnak (még egészen rosszul meghatározott helyzetekben is kitűnően működnek, bár ilyenkor gyakran meglepő eredményt adnak). Ezért érdemes az összes grafikus utasítást alaposan áttanulmányozni és grafikus ábrák rajzoltatása előtt az összes paramétert beállítani, nehogy egy korábbi beállítás az újabb futtatásban zavart okozzon. Az elmondottakból érezhető, hogy a gép grafikáját „teljesen" leírni reménytelen, megismerni azonban nem. A leírás ezt csak segíteni tudja, pl. azzal a megjegyzéssel, hogy minden grafikus utasítás a koordináták helyein relatív (a grafikus kurzor vagy pixel kurzor aktuális helyéhez viszonyított) koordinátákat is elfogad. Ha a koordináta explicit (kiírt) előjellel kezdődik, akkor az relatív, ellenkező esetben abszolút (a képernyő felosztásának megfelelő) koordinátaként értelmeződik *. Így pl. negatív abszolút koordinátát zárójelben vagy aritmetikai kifejezéssel adhatunk meg: (−1Ø) vagy Ø−1Ø alakban.

A grafikus utasítások 65 536-nál kisebb koordinátát fogadnak csak el. Az ehhez közeli értékek azonban negatív koordinátákat állítanak be. A következő relatív koordinátamegadással tehát látszólag átléphetjük a fent megadott értéket, azonban valójában nem ez történik. Ilyenkor a képernyő bal, ill. felső szélét lépjük át, az új koordináta már a megszokott (pozitív) tartományba kerül. (Itt feltétlenül meg kell jegyezni, hogy a grafikus utasítások is határesetekben igen furcsa jelenségeket idézhetnek elő, ezeket részletesen leírni nem érdemes. Ha az Olvasó ilyen problémába ütközik, akkor a gép „vallatásához" érdemes folyamodnia. Ehhez viszont segítséget nyújt a rendelkezésre álló számos leírás.)

Sokat vitatott a grafikus képernyő határolása. Az értelmező hibájaként emlegetik, hogy nem jelez hibát, ha a rajzzal „kilépünk" a grafikus képernyőről, pedig ez logikus következménye az osztott képernyős megjelenítésnek. Amint a képernyőkezelés ismertetésében már szó volt róla, az osztott képernyőknek kétféle alsó határa van; a megjelenítés határa és a grafikus képernyő határa. Indokolatlan, az átkapcsolhatóság miatt pedig célszerűtlen is lenne ezek valamelyikét kitüntetni.

Itt kell szólni az adatállományok kezeléséről is, még akkor is, ha ez nem tartozik szorosan a témánkhoz. Éppen ezért ne várja az Olvasó, hogy ezt az alábbi néhány mondatból megtanulhatja.

Elsőként az adatelem fogalmát kell bevezetni: ezen a logikailag együtt kezelt adatok csoportját értjük, ami persze a feldolgozás során változhat.

* Ezeken a helyeken polárkoordinátás relatív helymegadás is állhat. Ennek leírása a LOCATE kulcsszónál található.

Az adatállományok azonos logikai szerkezetű (vagy ilyenné alakított) adatelemekből állnak. Az adatállományok kezelésekor általában egyszerre csak egy ilyen adatelemmel dolgozunk. Ez leggyakrabban úgy történik, hogy valamelyik fizikai eszközt (lemezegység, nyomtató stb.) kijelölve megnyitjuk az adatállományt, ezzel jelezve, hogy azt hol (és milyen néven) kell keresnie, illetve létrehoznia a rendszernek. Ilyenkor az adatállományt egy számmal is ellátjuk, amely az adatállomány logikai száma lesz, programunkban ezzel nevezhetjük meg a kezelni kívánt adatállományt. (A logikai szám igazi jelentősége csak nagy adatkezelő programok futtatása és persze írása esetén érezhető igazán.) Végül az adatállományt az adatfeldolgozás után le kell zárni.

A COMMODORE gépeknek igen hatékony utasításkészlete van az adatállományok kezelésére. A C16 ebből a szempontból is kiemelkedő: a lemezegység kezelését különösen hatékonyan és kényelmesen oldották meg, és ez ráadásul a korábbi gépek lehetőségein túl biztosít többletszolgáltatást.

Az adatállományok kezelése természetesen sokkal nagyobb jelentőségű annál, semhogy ennyivel „megoldottnak" tarthassuk, ahhoz viszont elegendő, hogy a kulcsszavak leírását megértse az Olvasó.

Végül a leírás használatához fűzzünk néhány megjegyzést.

Gépünk minden aritmetikai értéken hajlandó végrehajtani az egész számmá való átalakítást: ilyenkor a szám egészrészének alsó 16 bitjét tartja meg, a túlcsordulást és a törtrészt levágja. A konverzió tehát nem vezet hibához, de nagy abszolútértékű számok esetén, ahol már az egészrész ábrázolása pontatlan, az átalakítás eredménye megbízhatatlan.

A leírásban nem szerepelnek a hibalehetőségek. Ennek oka az, hogy a hibák igen változatosak lehetnek: a hibás gépelésből adódó formai hibáktól kezdve, a megadott értékek hibáján át, a műveletek végrehajtási hibájáig. Ezekre a leírásban megadott határok alapján lehet következtetni. Ezt segíti a hibaüzenetek leírása is.

A leírásban nem szerepelnek a kulcsszavak szintaxisai (formális leírások), mert sok esetben — és éppen ezek lennének érdekesek — olyan bonyolult lenne a kulcsszó szintaktikájának megadása, hogy az gyakorlatilag használhatatlan lenne. A pontos szöveges leírás látszik könnyebben követhetőnek.

Nem találunk mintapéldát az egyes kulcsszavak alkalmazására sem. Azért nem, mert az ilyen példák általában csak programkörnyezetben értelmesek, így több más kulcsszót is érintenek, másrészt viszont a tipikus alkalmazásokra megadott minták a jó szöveges leíráshoz képest nem nyújthatnak újat. Alkalmazási példákat a mintaprogramokban találhat az Olvasó, de a BASIC-trükkök is segítik egy-egy kulcsszó alapos megismerését.

A leírásban * -gal jelölt kulcsszavakat a C64 is ismeri. A kulcsszó mellett találjuk annak rövidítését, majd ezt követi a kulcsszó rövid magyar jelentése.

ABS* aB NF abszolútérték

A zárójelbe írt argumentum abszolútértékét adja. Az argumentum aritmetikai, értéket adó kifejezés is lehet.

AND* aN M (logikai) és

A zárójelbe írt argumentum abszolútértékét adja. Az argumentum aritmetikai értéken bitenként végez „és" műveletet.

ASC * aS SNF ASCII kód

Az argumentumként megadott szöveg első karakterének ASCII kódját adja. Ha a szöveg üres (hossza nulla), akkor a függvény értéke Ø.

ATN * aT NF ArkuszTaNgens

Az argumentum arkusztangensét adja ívmértékben. A függvény értéke $-\pi/2$ és $\pi/2$ közé esik, az argumentum kifejezés is lehet.

AUTO aU P AUTOmatikus sorszámozás

A parancs után megadott paraméter az automatikus sorszámozás lépésköze lesz, miután egy (sorszámmal ellátott) programsor beírásával a funkciót aktivizáltuk. Ezek után minden újabb sor lezárásakor a következő sorszámot kiírja a gép és a kurzort a megfelelő helyre teszi. Üres sor beírásakor felfüggesztődik a funkció, de újabb érvényes (nem üres) programsor beírásával ismét aktivizálódik.

Kikapcsolni a paraméter nélkül megadott paranccsal lehet. Kikapcsolódik még a RUN hatására is, de pl. a NEW hatására nem.

Tört értékű paraméternek az egészrészét veszi, egynél kisebb paraméter esetén a paraméter nélküli funkció jön létre.

BACKUP bA U támogat, segít

Csak kétfutóműves lemezegységre használható, teljes lemezmásolást végző utasítás. A meghajtóegységek számait — amely Ø vagy 1 lehet — egy-egy D betű vezeti be, a forráslemez helyét az első szám adja meg, ezt a TO kulcsszó után a céllemez kijelölése követi. Ezután megadható a lemezegység száma, amit az előző résztől vessző, vagy az ON kulcsszó választ el és egy U betű vezet be. Ennek alapfeltételezés szerinti értéke 8.

Az utasítás hatására a céllemez megformázódik, így előző tartalma teljesen elvész.

Ha a lemezt önmgára másoltatjuk, akkor az mágnesesen frissítődik, tartalma változatlan marad. Még ilyen esetben sem adható ki egy futóművel rendelkező lemezegységre.

BOX bO U doboz

Az első paraméter által meghatározott színű téglalapot rajzol. E paraméter értéke nagyfelbontású grafikus üzemmódokban Ø vagy 1, többszínű üzemmódokban még 2 vagy 3 lehet. Ha elhagyjuk, akkor a gép 1-es értéket feltételez, de ilyenkor is kötelező kiírni az elválasztó vesszőt. Ezt a kötelezően megadandó egyik csúcs koordinátái követik, egymástól vesszővel elválasztva. Újabb vessző után adható meg az átellenes csúcs az előzőkhöz hasonlóan. Ha ez elmarad, akkor ezt a csúcsot a grafikus kurzor fogja kijelölni. A következő paraméter a téglalapnak a középpont körüli elforgatási szögét írja elő, az óramutató járásával egyirányban, fokban. Értéke legalább Ø, de 65 536-nál kisebb; ha elhagyjuk, akkor Ø-nak tekinti a gép. A következő paraméter 1-es értéke a téglalap kifestését, pontosabban kifestett téglalap rajzolását írja elő, Ø értéke mellett csak a téglalap határát rajzolja meg az első paraméter által előírt értékkel.

Ha második pontként a grafikus kurzor helyét akarjuk megadni, de elforgatási

szöget, ill. kitöltést is elő akarunk írni, akkor ezen pont koordinátáit egyetlen üres paraméterrel kell helyettesíteni, vagy relatív koordinátamegadást kell alkalmazni.

A téglalap rajzolása után a grafikus kurzor a másodikként megadott pont (elforgatás nélküli) helyére kerül.

Ha az utasítás a felsoroltnál több paramétert talál, akkor a rajzolás után hibát jelez. Szintaktikus hibát kapunk, ha az utasítás végén felesleges vesszők állnak.

CHAR chA U karakter

Pozicionált kiírást végző utasítás. Az első paramétere az írás színét jelöli ki, értéke Ø vagy 1, többszín üzemmódokban még 2 vagy 3 lehet, ha elhagyjuk, akkor 1 értelmeződik. Az első paramétert lezáró vesszőt az írás kezdőpozícióját kijelölő koordinátapár követi, egymástól és a következő szövegtől vesszővel elválasztva. Ezen három adat megadása kötelező. Az oszlop értéke Ø-tól 39-ig, a sor értéke Ø-tól 24-ig változhat, a szöveg üres is lehet. Az ötödik paraméterrel inverz kiírás állítható be: páros értékekre normál, páratlanokra inverz módú kiírást kapunk, de ezen paraméter egészrésze nem lehet Ø-nál kisebb és 255-nél nagyobb. Ha elhagyjuk, akkor a szöveg normál módban jelenik meg.

Ha a grafikus képernyőn a szöveg nem fér el a megadott sorban, akkor a folytatás a következő sorba kerül, de az utolsó sor vége után nem lépteti a képernyőt: a szöveg végét elhagyja.

A karakteres képernyőn végrehajtja a vezérlőkarakterek előírásait, de az ötödik paraméter inverz előírását figyelmen kívül hagyja. Grafikus üzemmódokban a vezérlőkarakterek hatástalanok; ilyenkor a listában is szereplő alakjuk inverzeként értelmeződnek.

Minden grafikus üzemmódban grafikus képernyőre vonatkozik az utasítás, így előfordul, hogy osztott képernyőre írva a karakteres képernyőrész „mögé" kerül a szöveg, tehát kijelzése elmarad.

Vegyük figyelembe, hogy többszínű üzemmódban a vízszintes felbontás felére csökken, így a kiírt szöveg — különösen inverz megjelenítés esetén — alig olvasható.

Mivel ez nem grafikus utasítás, így érthető, hogy relatív koordinátamegadást nem fogad el. A grafikus kurzort nem kezeli, de a karakteres kurzort igen, azt az utolsónak kiírt karakter utáni helyre teszi, tehát szükség esetén a képrnyőt is lépteti.

CHR$ * cH NSF CHaRacter: karakter

Az argumentumban megadott kódú karaktert adja, mint 1 hosszúságú szöveget. Az argumentum egészrésze legalább Ø és legfeljebb 255 lehet.

CIRCLE cI U kör

Törött vonal (poligon), kör, ellipszis és sokszög rajzolására szolgáló utasítás. Első paramétere Ø vagy 1 lehet nagyfelbontású grafikus üzemmódban, többszínű üzemmódban még 2 vagy 3 is. Ha elhagyjuk, 1 értékkel értelmeződik. A vesszőt követően a középpont koordinátái állnak, ezeket szintén vessző választja el egymástól. Csak együtt hagyhatók el, ilyenkor egy üres paraméterhelyet hagyva a grafikus kurzor jelöli ki a középpontot.

A következő paraméter a vízszintes irányú sugár (fél átmérő), megadása kötelező. Ezt a függőleges irányú sugár követi, amely elhagyható. Alapértelmezése a vízszintes sugár. Mindkét sugár csak 65 536-nál kisebb lehet, de negatív nem. A vízszintes irányú sugár 65 536-hoz közeli értékei negatív adatként értelmeződnek,

ilyenkor az „ívrajzolás" — egyéb adat hiányában — nem a legfelső, hanem az átellenes pontból indul. A függőleges adat 256-onként periodikus sugárméretet eredményez. A vesszőt követően a vonal középponti szögének kezdő, majd véghelyzetét adhatjuk meg fokokban. Értékük nem negatív, de 65 536-nál kisebb. Alapértelmezésük Ø, ill. 36Ø. Ezután megadható az alakzat elforgatásának szöge, szintén fokokban.

Ennek határai az előbbiekkel megegyeznek, alapértelmezése pedig Ø. Végül megadható a poligon szakaszaihoz tartozó középponti szög fokban, amely nem szükségképpen osztója a vonal középponti szögének. Ennek értéke legalább 1, de legfeljebb 255, alapértelmezése 2. Ha a vonalhoz tartozó középponti szög nem többszöröse a szakaszokhoz tartozóknak, akkor csak az utolsó szakasz középponti szöge lesz az előírttól eltérő. Ha a vonal középponti szöge kisebb az egy szakaszra megadottnál, akkor ezt nem veszi figyelembe; a kezdő és végszögek által meghatározott egyetlen szakaszt rajzolja meg.

Ha a felsorolásban szereplőnél több adatot adunk meg, akkor a gép rajzolás után ad hibajelzést.

Ha többszínű üzemmódban a függőleges sugár implicit megadású, akkor kört rajzol.

CLOSE * clO U bezárt, zárt, befejezés

Befejezi az adatállományok kezelését, törli ezek nyilvántartását a gépben. Természetesen az eszközön létrehozott adatállomány megmarad, de erről a program már nem tud a továbbiakban. A billentyűzetet és a képernyőt kivéve minden eszközt csak így kezel, eltekintve a program betöltésétől és kimentésétől.

Az utasítás utáni paraméter a lezárandó adatállomány logikai száma.

CLR * cL U CLeaR: tiszta, kitisztít, felszabadít

Törli a tárban levő változókat (a bejegyzésüket is), de a tárolt programot nem. Minden RUN és NEW hatására, valamint javításkor (programszerkesztéskor) is végrehajtódik. Törli a vermet is, és alaphelyzetbe állítja a DATA-mutatót.

GRAPHIC után megadva a grafikus képernyő számára lefoglalt tárterületet szabadítja fel. Ezt sem RUN, sem NEW hatására nem végzi el.

CMD * cM U CoMmanD: parancsol, vezérel, parancs, rendelkezés

Az utasításban megadott (előzőleg már megnyitott) eszközt jelöli ki elsődleges kimenő eszköznek. Ezután a nyomtató utasítások az így megadott eszközre vonatkoznak.

Az utasítás használatának és hatásának további tárgyalása túlságosan messze vezetne. Részletekről elsősorban az eszközök leírásában olvashatunk.

COLLECT colL U összegyűjt, összeszed

A lemez adatállományait rendező utasítás. Paraméterként megadható a meghajtó, valamint az egység száma, amit a D, ill. az U betű vezet be. Egymástól vessző, vagy az ON kulcsszó választja el őket. Alapértelmezésük 8, ill. Ø.

Hatására a kijelölt egységben levő lemez állományait újra megszervezi, és törli a hibás állományokat a tartalomjegyzékből.

Vigyázni kell arra, hogy ilyenkor az egyes adatok fizikai címei megváltozhatnak!

COLOR coL U szín

A képernyő kijelölt részének adott színűre állítását végzi.
Az első paraméter a zóna kijelölése:

Ø képernyőháttér,
1 előtér (karakterek, rajzolás),
2 többszín—1,
3 többszín—2,
4 képernyőkeret.

A második paraméter a színkód:

1 fekete,	5 bíbor,	9 narancs,	13 kékeszöld,
2 fehér,	6 zöld,	10 barna,	14 világoskék,
3 piros,	7 kék,	11 sárgászöld,	15 sötétkék,
4 türkiz,	8 sárga,	12 rózsaszín,	16 világoszöld.

A harmadik paraméter a fényerőt vezérli Ø-tól (sötét) 7-ig (világos). Ez a fekete színnél értelmetlen. A fényerő alapfeltételezés szerinti értéke 7.

CONT * cO P CONTinue: folytat(ódik)

A RUN/STOP-billentyűvel, vagy STOP- vagy END-utasítással leállított program folytatólagos indítását végzi. A folytatás közvetlenül a leállás helye utánról indul.
Csak parancsként használható!
A parancs nem hajtható végre, ha megállás után a programot szerkesztettük, vagy a változókat töröltük (ilyen esetekben ugyanis törlődik a verem is), továbbá ha a futás hiba miatt állt le, vagy parancsban hibát okoztunk.
A felfüggesztett program változói kiírhatók (lekérdezhetők), módosíthatók; a nyomonkövetés ki- és bekapcsolható.

COPY coP U másolat, másol

Adatállomány másolását végző utasítás. Első paraméterként megadható a meghajtó száma, ezt a másolandó állomány neve követi. A TO kulcsszó után a céllemezt tartalmazó meghajtó állhat, ezt (vagy a TO kulcsszót) a másolat (új) neve követi. Vesszővel vagy az ON kulcsszóval elválasztva megadható az egység száma. A meghajtó számát egy D betű, az egység számát egy U betű vezeti be. Ezek alapértelmezése 8, ill. Ø.
Ez az utasítás egyfutóműves lemezegységgel is használható.

COS * — NF koszinusz

Az argumentum (szög) koszinuszát adja. Az argumentumot ívmértékben (radiánban) értelmezi.

DATA * dA U adatok

Az utasítás után a programban felhasználandó adatokat adhatjuk meg. Az adatokat vessző választja el egymástól. A DATA utáni vessző, két egymást követő vessző, ill. az utasítást záró vessző üres szöveget, azaz Ø-t definiál. A szóközzel határolt, a

vesszőt, ill. kettőspontot tartalmazó szövegeket idézőjelek közé kell tenni. Adat nem tartalmazhat idézőjelet.

Parancsként megadva hatástalan, ugyanis „végrehajtáskor" a DATA-utasítást át kell lépni. Az így megadott adatokat azonban nem találja meg a rendszer, mert a DATA-mutató alaphelyzetben a program elején áll, a parancs értelmezése pedig a program előtti tárterületen történik, itt viszont nem keres adatot. Hibaállapotot tehát parancsként megadott DATA-utasítás nem okoz. (L. még **READ RESTORE**!)

DEC — SNF DECimal: tízes, tizedes

Az argumentumként megadott, legfeljebb négy hexadecimális számjegyből álló karaktersorozatnak megfelelő decimális értéket adja. (A hexadecimális számjegyek rendre a Ø,...,9 és A,...,F karakterek.)

DEF * dE U definíció

Függvényt definiáló utasítás. Használata: a „DEF FN függvénynév (formális változó) = kifejezés" utasítás után a bevezetett függvényre „FN függvénynév (argumentum)" alakban hivatkozhatunk. A függvénynév formátuma megegyezik a valós változók nevében használhatóval. A formális változó valós típusú, ilyen típusként kezeli az argumentumot is, és a függvény értéke is valós.

DELETE deL P töröl

A paraméterben megadott sorszámú, ill. a „—" jellel elválasztott sorszámhatárok által kijelölt sorokat (beleértve a határokat is) törli. Az elhagyott határ a program elejét illetve végét jelenti. A parancsban szereplő sorszámhivatkozások nem létező sorokra is vonatkozhatnak.

Csak parancsként használható!

DIM * dI U DIMension: kiterjedés, méret

Adattömbök méretének megadására szolgáló utasítás. A tömb neve után kell megadni az egyes dimenziók felső indexhatárait vesszővel elválasztva. Minden alsó indexhatár Ø. A tömbök lehetnek valós, egész vagy szöveg típusúak, a helyfoglalásuk ennek megfelelően különböző.

Egy DIM-utasításban több tömb is deklarálható. A dimenziók számát, ill. a tömb méretét valójában a tár nagysága korlátozza. A nem deklarált tömböt az első értékadáskor a rendszer automatikusan bevezeti minden (hivatkozott) dimenzióra 1Ø-es indexhatárral. Ha később a (tetszőleges módon) deklaráltnál nagyobb indexre próbálunk hivatkozni, akkor hibajelzést kapunk.

Egy már deklarált tömböt nem szabad újra dimenzionálni még ugyanolyan méretűre sem, mert ilyenkor a futás hibajelzéssel leáll. Ez korlátozza az egyébként dinamikus deklarálhatóságot, a felső indexhatárt ugyanis kifejezés is definiálhatja. (Ennek egy lehetséges megkerülését a BASIC-trükkök között találja az Olvasó, a 9. fejezetben.)

A CLEAR nem a tömbelemek értékeit nullázza, hanem törli a tömb nyilvántartását, így azt ismételten dimenzionálni lehet, ill. kell.

DIRECTORY diR P címtár (telefonkönyv)

A lemez tartalomjegyzékét a képernyőre listázza. Megadható a meghajtó száma, amit D betű vezet be, ezt követheti az egységszám, amit határolójelként vessző vagy

az ON kulcsszó előz meg, majd egy U betű vezet be. Ezután egy szövegkifejezés állhat, ami a listázás kulcsa lesz.

Az f_3-billentyűvel is elérhető.

DLOAD dL U Disc LOAD: betöltés lemezről

A parancs segítségével lemezről tölthetünk be programot. Ehhez kötelezően meg kell adnunk a program nevét, vagy a nevet (az adott lemezen) egyértelműen azonosító karakterláncot. Ez utóbbi tartalmazhat helyettesítő karaktereket is, amiről részletesen a lemezegység kézikönyvéből tájékozódhat az Olvasó. A név szövegkifejezés is lehet, pl. egy előzőleg értelmezett szövegváltozó, de azt zárójelbe kell tenni. A név üres szöveg nem lehet. Megadható még vesszővel elválasztva a meghajtó száma, ezt D betű vezeti be, továbbá vessző vagy az ON kulcsszó után az egység száma, amelyet U betű vezet be. Ezek alapértelmezése Ø, ill. 8. Programból is használható. (L. még **LOAD**!) Az f_2-billentyűvel is elérhető.

DO — U végrehajt, elvégez

Tetszőleges helyen — akár több helyen és feltétellel — tesztelő ciklus szervezésére szolgáló utasítás.

A ciklust a DO-utasítás nyitja meg; ez tartalmazza az elöl (is) tesztelő ciklus megfelelő ciklusfeltételét, amit az UNTIL vagy a WHILE kulcsszó vezet be. (L. még **EXIT, LOOP, UNTIL, WHILE**!)

Legfeljebb 39 DO—LOOP ciklus ágyazható egymásba. A GOSUB-bal és a FOR—NEXT-ciklussal közös vermet használ.

DRAW dR U rajzol, vonalat (ki)húz

Pontok, ill. egyenes szakaszok rajzolására használható utasítás. Első paramétere a színt jelöli ki, értéke Ø vagy 1 lehet nagyfelbontású grafika esetén, többszínű üzemmódban 2 vagy 3 is lehet. Alapértelmezése 1.

Vessző után a kezdőpont koordinátái állhatnak egymástól vesszővel elválasztva. Ha ez elmarad, akkor a kezdőpontot a grafikus kurzor jelöli ki. Megadhatók még a végpont koordinátái, amelyet a TO kulcsszó vezet be. Ha ezek elmaradnak, akkor a végpont a kezdőpontba esik. Ezzel egy szakasz (esetleg pont) megrajzolását írjuk elő, ennek végrehajtása során a grafikus kurzor a végpontba kerül.

Egy DRAW-utasításban tetszőlegesen sok szakaszmegadás szerepelhet, ezek nem szükségképpen összefüggők, és relatív koordinátamegadást is tartalmazhatnak. A koordináták természetesen csak párosával hagyhatók el.

DSAVE dS U Disc SAVE: (ki)mentés lemezre

A tárolt programot a kulcsszó után megadott névvel a lemezre menti. A nevet mindig meg kell adni, de ez szövegkifejezéssel is történhet. Megadható még vessző után a meghajtó száma, amit D betű vezet be, továbbá vessző vagy az ON kulcsszó után az egységszám, amit U betű vezet be. Ezek alapértelmezése Ø, ill. 8.

Programból is használható; ilyenkor a kimentés után a program tovább fut.

Az f_5-billentyűvel is elérhető.

ELSE eL U,K különben, egyébként

Az IF-utasítás hamis ágának kulcsszava, a THEN (igaz) ágat követi.

Az IF-utasítás végrehajtásakor a vezérlés az ELSE után álló utasításra adódik,

ha a feltétel értéke hamis (vagy Ø). Az ELSE után közvetlenül sorszám is állhat, azaz vezérlésátadáskor a GOTO kulcsszó elhagyható. Az ELSE előtt, azaz a THEN ág utolsó utasítása után ki kell írni az utasítást záró kettőspontot, különben az igaz ág végrehajtásakor hibát kapunk. A teljes hamis ág el is hagyható.

Aktív IF-utasítás nélkül az ELSE-utasítás is ugyanúgy hajtódik végre, mint a REM.

END* eN U vég(e)

Az utasítás végrehajtásával a program futása leáll, a gép parancsüzemmódba kerül és bejelentkezik. Ilyenkor — ha van értelme — a program a CONT-paranccsal folytatható, ugyanis a változók nem törlődnek.

Elsősorban akkor használjuk, ha a program logikai és fizikai vége nem esik egybe, különben használata felesleges (implicit programvég).

Jól alkalmazható programbelövés során is, ugyanis a bejelentkezésen kívül mást nem ír a képernyőre, így az — folytatáskor — kevésbé lesz „szemetes".

ERR$ eR NSF ERRor: hiba

Az argumentumban megadott hibakód (szöveges) hibaüzenetét adja. Az argumentum legalább 1, de 37-nél csak kisebb lehet.

A fellépett hiba kódját az ER, a hiba sorának sorszámát az EL fenntartott BASIC-változók tartalmazzák.

EXIT exI U kijárat

Magban tesztelő DO—LOOP-ciklus szervezését teszi lehetővé ez az utasítás. Végrehajtásakor a logikailag utolsó ciklust záró, azaz a programszövegben soronkövetkező LOOP-utasítást követő utasításra kerül a vezérlés és törlődik a ciklus nyilvántartása.

Az ilyen ciklusokban nagyon célszerű elkerülni a vezérlésátadásokat, mert ezek igen bonyolult (vezérlésátadási, ill. veremkezelési) hibákat okozhatnak.

EXP* eX NF EXPonent: (hatvány)kitevő

Az „e" szám (a természetes alapú logaritmus alapjának) hatványát adja. A kitevő a függvény argumentuma. (Gondoljunk arra, hogy az exponenciális függvény növekvő argumentumok esetén gyorsan nő, és túlcsordul, csökkenő argumentumok esetén gyorsan tart Ø-hoz, így az eredmény gépi Ø lesz.)

FN* — K FuNction: függvény

A programban definiált függvényre való hivatkozás kulcsszava. Ezt a függvény neve, majd zárójelben az argumentuma követi. (L. még **DEF**!)

FOR* fO U részére, számára, ...

A BASIC általánosan elterjedt ciklusszervezésének kulcsszava: az utána következő értékadással kijelöli a ciklusváltozót és beállítja ennek kezdőértékét. Ezt a végérték és — ha kell — a lépésköz megadása követi. A FOR-utasítás végzi a ciklus bejegyzését.

A FOR—NEXT-ciklus hátul tesztel. Legfeljebb 1Ø ilyen ciklus ágyazható egymásba, de a GOSUB-bal és a DO—LOOP-ciklussal közös vermet használ. (L. még **TO, STEP, NEXT**!)

FRE * fR NF FREe: szabad, mentes

A szabad tárterület nagyságát szolgáltató függvény. Álargumentumot használ; az argumentum meglétét ellenőrzi, de típusát nem. Végrehajtáskor a szövegterületet tömöríti (törli a már nem aktuális láncokat), és így nézi meg a szabad byte-ok számát.

A grfikus tár foglaltnak minősül. (L. még **GRAPHIC**!)

GET * gE U kap, szerez, eljut, jön

Karakter bevitelére szolgáló utasítás, amely az utasításban megadott számú (megnyitott) állományból olvas be a változó(k)ba egy-egy karaktert. Az állomány számát egy kettőskereszt (#) vezeti be, és a GET kulcsszó után, a változó(lista) előtt áll. Állománykijelölés nélkül az utasítás a billentyűzetre vonatkozik: végrehajtáskor az ún. billentyűzet-puffert kezdi olvasni az értelmező. Ha ez üres, akkor az utasítás után írt numerikus változó értéke Ø, a szövegváltozó értéke az üres szöveg lesz. (A billentyűzet-puffer egy 1Ø byte-os tárterület, amely mint csőtár (FIFO) működik. Ha ez megtelt, akkor a további billentyűleütéseket a rendszer nem veszi figyelembe. A program befejezése utáni bejelentkezéskor a rendszer kiüríti a puffert: a legfeljebb tíz karaktert a képernyőre írja, majd a kurzort a következő pozícióra állítja, hacsak a szöveg mást nem ír elő, pl. a sorzárás miatt.)

Ha a GET után és a változó (lista) előtt a KEY kulcsszót is megadjuk, akkor a gép üres billentyűzet-puffer esetén a következő billentyű leütéséig vár minden lista-elembe való beolvasásnál.

Ha számváltozóba nem számkaraktert olvastatunk, akkor típuskeverési hibát kapunk.

Csak utasításként használható.

GO * — U megy (menj!)

Csak a TO kulcsszó előtt használható (közöttük csak szóközök állhatnak). Együttes hatásuk csaknem megegyezik a GOTO kulcsszóéval, végrehajtásuk azonban valamivel lassabb, tárolásuk legalább két byte-tal többet igényel. A GO TO alakkal nem helyettesíthető a THEN, és az ON utasításban sem állhat. (L. még **GOTO**!)

GOSUB * goS U menj a szubrutinra!

Az utasítás hatására elmentődik (a veremtárba) az utasítás végének címe és az aktuális sorszám. Ezután a vezérlés az utasításban megadott sorszámú sorra adódik. A RETURN hatására tér vissza a veremben őrzött címre.

Legfeljebb 39 szubrutinhívás ágyazható egymásba, de a FOR—NEXT- és a DO—LOOP-ciklussal közös vermet használ.

GOTO * gO U GO TO: menj valamihez!

Feltétlen vezérlésátadást hajt végre a kulcsszó után megadott sorszámú sorra. A hivatkozott sorszámnak léteznie kell a programban.

Az IF-utasításban, ha csak vezérlésátadás történik az igaz ágban, akkor a THEN kulcsszó helyett közvetlenül használható.

Ha a programot (parancs üzemmódban) GOTO-val indítjuk, akkor a változók addigi értékei nem törlődnek.

A GO TO alakkal nem egyenértékű. (L. még **GO**!)

GRAPHIC gR U grafikus

A képernyő üzemmódját beállító utasítás. Az első paraméter az üzemmódot írja elő, megadása kötelező. Megengedett értékeit és ezek jelentését az alábbi felsorolás tartalmazza.

Ø karakteres képernyő,
1 nagyfelbontású grafika,
2 nagyfelbontású grafika osztott megjelenítéssel,
3 többszínű grafika,
4 többszínű grafika osztott megjelenítéssel.

A második paraméterrel a törlés írható elő: ha értéke 1, akkor törlődik a bekapcsolódó képernyő, ha Ø, vagy a paraméter elmarad, akkor nem törlődik.

Ha a paraméter(ek) helyett a CLR kulcsszót adjuk meg, akkor a grafikus képernyő tárterületét felszabadítja a BASIC számára. Ezt önmagában még a NEW-parancs sem végzi el.

GSHAPE gS U Graphic SHAPE: grafikus alak, forma

Egy előzőleg szövegváltozóba töltött grafikus képtartalmat jelenít meg a képernyőn. Ügyeljünk arra, hogy a képernyő pillanatnyi állapota (nagyfelbontású vagy többszínű grafikus képernyő) megfeleljen a megjelenítendő ábrának, ellenkező esetben ugyanis meghatározatlan helyzet jön létre.

Az utasítás első paramétere a képtartalmat tároló szövegváltozó, amelyet kötelező megadni. Megadhatók a megjelenítendő téglalap bal felső csúcsának a koordinátái. Ezek együtt hagyhatók el, ekkor a csúcsot a grafikus kurzor jelöli ki. Harmadik paraméterenként az üzemmód írható elő. Ennek jelentése nagyfelbontású grafika esetén:

Ø a téglalapot az adott helyre másolja;
1 a téglalap inverzét az adott helyre másolja;
2 a kijelölt terület azon pontjait is írás színűre állítja, amelyek az elmentett alakzaton ilyenek;
3 a kijelölt terület azon pontjain hagyja meg az írás színét, amelyek az elmentett alakzaton is ilyenek;
4 ellenkező színűre váltja a kijelölt terület azon pontjait, amelyek az elmentett alakzat írás színű pontjainak felelnek meg.

Többszínű üzemmódban az utasítás üzemmódjai közül csak a Ø üzemmód használata egyszerű. A többi üzemmód igen bonyolult helyzetekre vezethet, melyeket nagy számuk miatt nehézkes áttekinteni. (Ehhez nyújt némi segítséget az egyik mintaprogram.)

Az üzemmód alapfeltételezés szerinti értéke mindegyik esetben Ø. (L. még **SSHAPE**!)

HEADER heA U fejrész, főcím, valamit fejrésszel ellátó

Lemez formázását végző utasítás. Első paraméterként a lemez (új) nevét kell megadni. Ennek hossza legfeljebb 16 karakter lehet, különben hibajelzést kapunk. Ha ezt egy szövegváltozóval adjuk meg, akkor a változó nevét zárójelbe kell tenni. Ezután meg kell adni a meghajtó számát, amit a névtől vessző választ el és egy D betű vezet be. Vesszővel elválasztva megadható a lemez azonosítója, amit I betű vezet be és két karaktert tartalmaz. Ha elhagyjuk, akkor csak törli az előzőleg már formázott

lemezt, és új névvel látja el, de az azonosító változatlan marad és a lemez felosztását sem végzi el. Így érthető, hogy ilyenkor formázatlan lemez esetén lemezhibát jelez a gép. Végül vesszővel vagy az ON kulcsszóval elválasztva megadható a lemezegység száma. Ezt U betű vezeti be, alapértelmezése 8.

HELP heL U (meg)segít, támogat; segítség

Hibaállapotban kilistázza a hibát okozó sort, villogtatva azokat az utasításokat, amelyek már nem hajtódtak végre. Programból is végrehajtható; pl. a hibakezelő rutinban lévő HELP-utasítás végrehajtása után folytatódik a rutin és így a program is.

Az utasítást általában parancsüzemmódban használjuk a program javításakor. Ügyeljünk arra, hogy javítás (programszerkesztés) után a hibaállapot megszűnik, az utasítás ekkor már hatástalan, csakúgy, mint a hibakezelő rutinból való visszatérés után.

A HELP billentyűvel is elérhető.

HEX$ hE NSF hexadecimális

Az egész (számmá konvertált) argumentum négykarakteres hexadecimális alakját adja. A nem negatív argumentum csak 65 536-nál kisebb lehet.

IF * — U ha

A kulcsszót követő logikai vagy aritmetikai kifejezés értékétől függő elágazást tesz lehetővé. Az aritmetikai kifejezés hamisnak minősül, ha értéke Ø, különben igaznak tekinti az értelmező. (L. még **ELSE, GOTO, THEN**!)

INPUT * — U bemenet(i adat)

A csak programban használható utasítás futás közbeni adatbevitelre szolgál. A kulcsszó után üzenet állhat, amely csak idézőjelek között megadott szövegkonstans lehet. Ezt elhatároló jelként pontosvessző zárja le. A kulcsszót, ill. az üzenet utáni pontosvesszőt a nem üres változólista követi.

Végrehajtáskor a gép kiírja az üzenetet — ha van —, az aktuális pozícióra egy kérdőjelet, majd egy szóközt ír és a következő pozícióra teszi a kurzort, ezután a billentyűzetről adatot vár. Az adatbevitel a sorzárásig tart. Ezután a bevitt adatot, ill. az egymástól vesszővel vagy kettősponttal elválasztott adatokat az utasításban felsorolt változókba tölti.

Több változó bevitelekor ügyeljünk arra, hogy a változók és az adatok típusa illeszthető legyen! Ha a szükségesnél kevesebb adatot adunk meg a végrehajtáskor, akkor a következő sor elején két egymást követő kérdőjel jelenik meg, és a rendszer további adatbevitelre vár. Ha a listában kevesebb változó áll, mint a bevitt adatok száma, akkor „? EXTRA IGNORED" hibaüzenet kiírása után a program tovább fut és a felesleges adatokat figyelmen kívül hagyja.

Érdemes szólni a számkarakterek kezeléséről: ezeknek bármely, a nyomtatási kép formátumának megfelelő sorozatát értelmezi a gép. Ha a mantissza elmarad, akkor a számváltozó értéke Ø lesz. A mantissza „helyettesíthető" egyetlen előjellel vagy tizedesponttal, a karakterisztika pedig csak egyetlen előjellel. (A karakterisztikában ugyanis csak előjeles egész érték állhat.)

Ha vesszőt vagy kettőspontot tartalmazó szöveget is szeretnénk beolvastatni az utasítással, akkor azt idézőjellel kell kezdeni, ez ugyanis jelzi az értelmező számára, hogy a további karaktereket egyetlen szövegként kell kezelnie. A szöveg azonban nem tartalmazhat idézőjelet, legfeljebb csak a sor végén, de ez elhagyható.

Az alábbi utasítássorozattal kényelmesebbé tehetjük az idézőjeles adatbevitelt.

POKE 239,1: POKE 1319, 34: INPUT" üzenet"; A$

(A csak programban megadható utasítássorozat először a billentyűzet-puffer aktuális hosszát 1-re állítja, majd az idézőjel kódját beteszi a pufferbe. Az input végrehajtásakor kiírja az üzenetet, a kérdőjelet és a szóközt. Mivel a billentyűzet-puffer nem üres, így a tartalmát átírja az I/O pufferbe, és természetesen a képernyőn is megjelenteti. Ezután kezdődhet az inputsor bevitele.)

A fenti trükk alapján könnyen látható, hogy minden INPUT-utasítás előtt „illene" kiüríteni a billentyűzet-puffert, mert abban esetleg „teljes" inputsor lehet (sorzárással!), és a felhasználó számára érthetetlen módon kezd viselkedni a gép. Az ürítést pl. a FOR I = Ø TO 9: GET A$: NEXT-ciklussal, vagy — egyszerűbben, de gépspecifikusan — a POKE 239, Ø utasítással végezhetjük el.

Ügyelnünk kell arra, hogy az INPUT-utasítás először egy kiírást hajt végre, amely az elsődleges kimeneti állományba ír, a bevitel azonban a billentyűzet-pufferből, a képernyőn megjelenítő karakterek alapján történik. (L. **CMD**!)

Figyelem! Az I/O puffer csak 88 byte hosszú!

INPUT‡ * iN U bemenet(i adat)

Az INPUT-utasításhoz nagyon hasonlóan működik, a perifériákról történő beolvasásra szolgál.

A kulcsszót az olvasandó (és előzőleg már megnyitott) adatállomány logikai száma, majd vessző után a változólista követi.

A speciális karakterek jelentése megegyezik az INPUT-nál leírtakkal, azonban az elválasztójelek miatt itt általában nem hagy el adatot a rendszer.

Ügyeljünk arra, hogy az I/O puffer hossza itt is csak 88 byte!

INSTR inS TF IN STRing: láncban

A függvény két szövegargumentumot vár, és megadható még egy számargumentum. Értéke a második szövegnek az elsőben való első előfordulásának kezdőpozíciója. Ha a harmadik argumentumot is megadjuk, akkor a keresés az általa meghatározott pozíción kezdődik. (Ennek alapértelmezése tehát 1.)

Ha a második szöveg nem része az elsőnek (az adott kezdőpozíciótól számítva), ill. ha valamelyik szöveg üres, akkor a függvény értéke Ø. A számargumentum értéke legalább 1, de kisebb, mint 256. (Nyilvánvaló, hogy ha kezdőpozíciónak az első szöveg hosszánál nagyobb számot adunk meg, akkor a függvény értéke Ø lesz.)

INT * — NF INTeger: egész(rész)

Az argumentum — valós ábrázolású — egészrészét adja. (Azt a legnagyobb egész számot, ami az argumentumnál nem nagyobb.)

JOY jO NF JOY-stick: botkormány, kormányrúd

Az argumentum által kijelölt botkormány állapotát adja. Az argumentum (egészrésze) csak 1 vagy 2 lehet.

A függvény értéke a botkormány alapállapotában Ø. Nyolc irányt érzékel, melyekhez 1-től 8-ig rendel értékeket az előre (fel)helyzettől az óramutató járásával egyező irányban. Ha a „TŰZ" gomb is le van nyomva, akkor a felsorolt értékekhez 128 adódik.

KEY kE U,K kulcs, billentyű

Paraméter nélkül megadva kilistázza a funkcióbillentyűk aktuális jelentését. Használható a funkcióbillentyűk átértelmezésére is. Ilyenkor első paraméterként a megfelelő billentyű számát kell megadni, amelynek értéke 1-től 8-ig változhat, majd vessző után az új jelentés áll, amit szövegkifejezéssel is megadhatunk.

Ha a GET-utasítás után adjuk meg, akkor — üres billentyűzet-puffer esetén — a gép billentyű leütésére vár mindegyik listaelembe való beolvasáskor. (L. **GET**!)

LEFT$ * leF TF bal oldal

A függvény az első (szöveg-)argumentumnak a második (szám-)argumentum által meghatározott hosszúságú bal oldali részét adja. Ha az argumentumszöveg rövidebb, mint az előírt hossz, akkor az argumentum (eredeti hosszúsággal) lesz a függvényérték. A számargumentum egészrésze legalább Ø, de legfeljebb 255 lehet.

LEN * — SNF LENgth: hossz(úság)

Az argumentumként megadott szöveg hosszát adja, beleértve a vezérlőkaraktereket és a szóközt is.

LET * lJ U hagy, enged (legyen!)

Az értékadás kulcsszava, amelyet az értelmező nem követel meg. Az értékadó egyenlőség bal oldalán egyetlen változónév, jobb oldalán egy, a változó típusának megfelelő aritmetikai, logikai vagy szövegkifejezés áll.

Ügyelni kell a változók és a kifejezések típusára, a túlcsordulásra és a szövegkifejezések hosszára.

A MID$ függvény speciális esetként a bal oldalon is állhat. (L. **MID$**!)

LIST * lJ U lajstrom, (program)lista

A „—" jellel elválasztott sorszámhatárok által meghatározott programsorokat listázza ki az elsődleges kimeneti eszközre. A hiányzó sorszámhatár(ok) a program elejét, ill. végét jelenti(k). Ha egyetlen sorszámot adunk meg, akkor a hivatkozott sort írja ki, ha ilyen sorszámú sor van a programban, ha nincs, akkor a kiírás elmarad.

A megadott sorszámoknak nem kell feltétlenül létezniük a programban.

Az f_7-billentyűvel is elérhető.

LOAD * lO U (meg)tölt, töltés

Az utasítással kazettáról vagy lemezről tölthetünk be programot. A kulcsszó után álló szövegkifejezés a betöltendő program nevét írja elő, amely csak kazettás egységnél hagyható el, amikor is a szalagon soron következő program töltődik be. Ha a lemez tartalomjegyzékében elsőként szereplő programot akarjuk betölteni, akkor idézőjelek között egy csillagot (ún. jolly-karakter) adjunk meg. A csillag kazettás egységen nem használható úgy, mint a lemezegységen.

A programnév után megadható az egységszám, aminek alapfeltételezés szerinti 1-es értéke a kazettás egységre vonatkozik. Lemezegység esetén az egység száma adandó meg.

Harmadik paraméterként a betöltési cím jelzője adható meg. Ennek Ø értéke a BASIC területre való betöltést ír elő; ez a jelző alapértelmezése. Ha a jelző értéke 1, akkor a betöltés a tárolt címen kezdődik. (Ez utóbbi a gépi kódú programok betöltésekor fontos.)

Programban megadva betölti az előírt nevű programot és azt az elejéről elindítja. Ezzel a lehetőséggel azonban óvatosan éljünk, ugyanis ilyenkor a változóterületet nem mozgatja, amely tehát felülíródhat. Ez a programláncolás lehetőségét súlyosan korlátozza. (L. még **DLOAD**!)

LOCATE loC U elhelyez, a helyét megállapítja

A grafikus kurzort elhelyező utasítás, amely mindig két paramétert vár. Ha a paramétereket vessző választja el egymástól, akkor azok koordinátaként értelmeződnek. Abszolút és relatív koordináta egyaránt használható, akár vegyesen is. Ha a paraméterek között pontosvessző áll, akkor az első paraméter előjeles (!) távolságként értelmeződik, a második pedig fokban megadott irányszögként, amely a képernyőn felfelé mutató iránytól az óramutató járása irányában nő.

Pl. a LOCATE 20; 90 és a LOCATE −20; 270 utasítások hatására ugyanaz történik: a grafikus kurzor 20 egységgel jobbra mozdul el.

LOG* — NF logaritmus

Az argumentum természetes alapú logaritmusát adó függvény. Az argumentum kifejezés is lehet, de csak pozitív értékű.

LOOP loO U hurok, karika

A DO-utasítással megnyitott ciklust lezáró utasítás. Hátul tesztelő ciklusnál a tesztelést közvetlenül a LOOP után álló UNTIL vagy WHILE kulcsszóval szervezhetjük meg.

Minden DO-utasításhoz pontosan egy LOOP tartozik, ami akár THEN, akár ELSE ágban, akár GOTO után is állhat.

(L. még **DO, EXIT, UNTIL, WHILE**!)

MID$* mI TF közép(ső), középen levő

A függvény argumentumaként egy szöveget és egy vagy két számot vár. A függvény értéke a megadott szövegnek az első szám által kijelölt pozíciótól kezdődő, a második szám által meghatározott hosszúságú része, ha a szöveg elég hosszú. Ha a szöveg rövidebb, akkor az eredmény az adott pozíciótól a szöveg végéig tartó rész, vagy üres szöveg, ha már az első szám is nagyobb, mint az adott szöveg hossza. Ha a második számot elhagyjuk, akkor 255-nek értelmeződik.

A függvény valamely értékadás bal oldalán is használható. Ilyenkor az argumentumban megadott szövegváltozó egy részét felülírja a jobb oldalon megadott szöveggel, ennek azonban el kell „férnie" az argumentumszövegben az adott pozíciótól kezdődően. A második számargumentumot ilyenkor nem használja, az elhagyható.

Az első számargumentum legalább 1, a második legalább 0, és mindegyikük kisebb 256-nál. FIGYELEM! A függvény az argumentumot akkor is megvizsgálja, ha nem használja fel!

MONITOR mO P (meg)figyelő (berendezés)

A paranccsal a — gépi kódú programozást segítő — beépített monitor-program aktivizálható. A monitorból visszatér a BASIC-be, ha egyetlen X-et tartalmazó sort adunk be.

Programból kiadva is parancs üzemmódba tér vissza, de a változók értékei megmaradnak.

NEW * — P új

A tárolt BASIC-programot és a változókat törlő parancs. A törlés csak logikai: a programterület elejére beírja, hogy ott a program vége, és a következő byte-ra állítja a változótábla összes mutatóját, amivel a változókat törli.

Gondoljunk arra, hogy a NEW számos jellemzőt nem állít át, így továbbra is érvényben maradnak az AUTO-parancs és a grafikus beállítások hatásai is.

A NEW programból is végrehajtható, de ilyenkor is parancsüzemmódba tér vissza a rendszer.

NEXT * nE U,K következő, legközelebbi

A FOR-utasítással megnyitott ciklust lezáró utasítás. Végrehajtásakor megnöveli a ciklusváltozó értékét, majd ha az a végértéket még nem lépte túl, akkor visszatér a ciklusmagra, különben törli a ciklus bejegyzését és a következő utasításra tér.

Ha a NEXT után nem adunk meg ciklusváltozót, akkor a legutoljára megnyitott, még érvényes (logikailag utolsó) ciklust zárja le. Ha a ciklusváltozót is megadjuk, akkor az adott ciklust zárja le, és minden ennél később bejegyzett ciklust töröl.

A FOR—NEXT-ciklus több helyen is lezárható, mivel hátul tesztel.

A ciklusváltozó kiírása a cikluskezelés (növelés, összehasonlítás, vzérlésátadás) időtartamát mintegy negyedével megnöveli. (L. még **FOR, TO, STEP**!)

A NEXT kulcsszóként is szerepelhet; a RESUME után a hibakezelő rutinból való visszatérés helyét jelöli ki. (L. **RESUME**!)

NOT * nO M nem (tagadás)

A kulcsszó utáni logikai érték vagy egész (számmá konvertált) érték (bitenkénti) invertálását végző művelet.

ON * — U,K be (van kapcsolva)

Többirányú elágazást szervező utasítás.

A kulcsszót követő (aritmetikai) kifejezés egész részének megfelelő sorszámú sorszámhivatkozásra adja a vezérlést, amely a kifejezés után álló kulcsszótól (GOTO vagy GOSUB) függően ugrás vagy szubrutinhívás lehet. A sorszámlista a GOTO, ill. a GOSUB után következik. Ha a kifejezés egészrésze — amely Ø-nál kisebb és 255-nél nagyobb nem lehet — Ø, vagy nagyobb, mint a felsorolt sorszámok száma, akkor a következő utasításra tér.

Az ON-utasításban sem használható a GO TO alak. Kulcsszóként a lemezes utasítások egységkijelölésének határolójeleként használható.

OPEN * oP U nyílik, nyitott, nyílt

Adatállományok megnyitását végző utasítás.

Hatására a rendszer nyilvántartásba veszi a kijelölt adatállományt az első paraméter által meghatározott logikai számmal. Az állományra a későbbiek során ezzel lehet hivatkozni. Ezután az eszköz száma következhet (amelyen az állománykezelés történik), ha elmarad, akkor az a kazettás egységre vonatkozó 1-es értékkel értelmeződik.

A következőnek megadható paraméter kazettás egység esetén a kezelés módját, lemezegységnél a kommunikációs csatornát jelöli ki. (Ezek olyan COMMODORE-sajátosságok, amelyek részletes ismertetése túlságosan messze vezetne, így elsősor-

ban a perifériák kezelési utasításaira és a COMMODORE 64 leírásaira támaszkodhat az Olvasó. Természetesen az állományok szerkezetének és szervezésének alapjait feltétlenül ismerni kell!) Megadható még az állomány neve — és néhány egyéb jellemzője is — a negyedik paraméterben. Ez azonban már csak komoly állománykezelés esetén szükséges, egyszerű perifériakezelési műveletekhez — pl. nyomtatóra való íráshoz — az első két páraméter használata és ismerete elegendő.

OR * — M (logikai) vagy

A kulcsszó előtt és után álló logikai értékek, ill. egész (számmá konvertált) értéken bitenkénti „vagy" műveletet végez.

PAINT pA U fest; festék

Vonalakkal zártan határolt grafikus területek színezését végzi. Elsőként a szín adható meg, amely nagyfelbontású grafika esetén Ø vagy 1, többszínű grafika esetén még 2 vagy 3 lehet. Alapértelmezése 1. Ezután a festendő terület egy pontjának koordinátái állhatnak, melyeket elhagyva, a pontot a grafikus kurzor jelöli ki. Végül az üzemmód megadása állhat, amelynek az alapértelmezés szerinti Ø értéke mellett az utasításban megjelölt színnel (zártan) határolt területet festi ki, ha értéke 1, akkor a nem háttér színű pontokig folytatódik a festés.

Ha a kezdőpont (az üzemmódnak, ill. színnek megfelelő) határra esik, akkor nem fest.

(Gondoljuk végig, hogy a háttérszínnel való festés esetén — különösen a már amúgyis bonyolult, többszínű ábrán — nehezen érthető színezések is történhetnek!)

PEEK * pE NF (ki)néz valahová

Az argumentumban megadott tárcím tartalmát szolgáltatja. Az argumentum csak nem negatív és 65 536-nál kisebb szám lehet, míg a byte tartalma csak Ø,...,255 lehet.

POKE * pO U döf, beszúr

Az első paraméter által meghatározott tárcímre a második paraméter értékét tölti. A paraméterek egészrésze Ø,...,65 535, ill. Ø,...,255 lehet.

POS * — NF POSition: helyzet, állás

A függvény a kurzornak az aktuális sorban való helyét adja, értéke tehát Ø,...,39 lehet. Az argumentum típusa is, értéke is közömbös, kiírni azonban kötelező (ál-argumentum).

PRINT * ? U nyomat; nyomtatás; (ki)nyomtat

Az utasítás után álló tétellistát írja ki az elsődleges kimeneti adatállományba (a hozzá tartozó eszközre). Ez — ha másként nem intézkedünk — a képernyő. (L. még **CMD, OPEN, CLOSE**!)

A tétellista elemei az alábbiak lehetnek:

- ☐ konstansok, változók, kifejezések, amelyek egyaránt lehetnek aritmetikai, logikai és szöveg típusúak;
- ☐ a TAB és az SPC pozicionáló függvények (l. ott);

□ formátumvezérlés, ami csak a lista első elemeként állhat (részletesebben l. USING);
□ a vessző, a pontosvessző és esetenként a szóköz, mint elhatároló jelek.

A vessző hatására a kurzor a listaelem kinyomtatása után a következő zóna elejére megy (a C16 négy azonos hosszúságú zónát használ a képernyő minden sorában), míg a pontosvessző hatására a kurzor a nyomtatást követő pozíción marad. A nyomtatandó listaelem üres is lehet, így egymás után is állhatnak elhatároló jelek.

A pontosvessző elhagyható, ha az egymást követő listaelemek balról jobbra haladva megkülönböztethetők. Így idézőjel, zárójel és a változó típusjele után nem kell kiírni a pontosvesszőt. Ha azonban az AB nevű valós változó után I darab szóközt akarunk íratni, akkor azt a PRINTAB; SPC (I) vagy a PRINTAB SPC (I) alakban kell megadni, mert a PRINTABSPC (I) szintaktikus hibát eredményez a PRINT után álló ABS betűcsoport miatt. A szóköz tehát nem speciálisan a nyomtatás elhatároló jele, hanem mint egyetemes elhatároló jel használható.

A lista üres is lehet, ilyenkor — nyomtatás nélkül — a következő sor elejére áll a kurzor.

Ha nem a képernyő az elsődleges kimeneti eszköz, akkor az elmondottak csak az eszköz szabta korlátok között érvényesek, ilyenkor az eszköz leírását kell alapul venni.

PRINT# * pR U nyomat; nyomtatás; (ki)nyomtat

Az utasítás segítségével bármely — előzőleg (írásra) megnyitott — adatállományba (a megfelelő eszközre) írathatunk. A kulcsszó után az állomány logikai száma áll, ezt egy vessző, majd a tétellista követi.

A PRINT‡ utasítás a logikai állományszám kezelésétől eltekintve gyakorlatilag megegyezik a PRINT-utasítással, de ez utóbbi csak az elsődleges kimeneti állományba (eszközre) tud írni. (L. **PRINT**!)

PUDEF pU U Print Using DEFinition: PRINT USING definíció

A formátumvezérlés formátumkaraktereit átértelmező utasítás. A kulcsszó után (idézőjelek között) megadott (nem üres) szöveg első karaktere a szóközök, második karaktere a — számjegyek csoportosítását végző — vesszők, a harmadik pedig a tizedespont helyett jelenik meg. A negyedik karakter a dollárjel új értelmezése, amely azonban csak akkor hatásos, ha a formátum előírásában — a dollárjel megengedett helyén — is az átértelmezett karakter áll. Az értelmező szöveg négynél több karaktert nem tartalmazhat, de kifejezés is lehet. Az utasítás formázott szövegnyomtatás esetén hatástalan.

RCLR NF Recall CoLoR: a „COLOR" visszahívása

Az argumentum által kijelölt képernyőtartomány színállapotát szolgáltató függvény. Az argumentum a COLOR utasítás első, a függvényérték a második paraméterének felel meg. Az argumentum egészrésze Ø-tól 4-ig változhat, az aktuális grafikus üzemmódtól függetlenül. (L. még **COLOR, RLUM**!)

RDOT rD NF Recall DOT: a pont (értékének) visszahívása

A grafikus kurzor állapotát adja az argumentum függvényében. Az argumentum értékei:

Ø x koordináta,
1 y koordináta,
2 szín.

READ * rE U (el)olvas

Az utasítás után álló változó(k)ba beolvassa a DATA-utasításokban megadott, soron következő adato(ka)t. Ha ezek elfogytak, vagy típusuk nem illeszthető a változó(k)éhoz, akkor a hibajelzést az olvasás sorára kapjuk. Ha az utasításban több változót (lista) adunk meg, akkor azokat vessző választja el egymástól. (L. még **DATA, RESTORE**!)

A változólista nem lehet üres.

REM * — U REMark: megjegyzés

A programsor kulcsszó utáni részét a rendszer nem értelmezi, egyszerűen átfut rajta. A megjegyzés a sor végéig tart.

A program tagolására, a blokkok megjelölésére, ill. a működés kommentálására bevezetett utasítás. A megjegyzések lekezelése lassítja a futást.

Parancsüzemmódban a REM a parancs végét jelenti.

RENAME reN U újranevezés, átnevezés

Első paraméterként a meghajtó száma adható meg, amelyet egy D betű vezet be és vessző zár le. Elhagyva, a meghajtó számaként Ø értelmeződik.

Kötelező megadni az állomány régi és új nevét, melyeket a TO kulcsszó választ el, és mindegyik szövegkifejezés is lehet. (A változókat tartalmazó kifejezéseket általában zárójelbe kell tenni.) Ha hibás állománynevet adunk meg (a régi név nem szerepel, ill. az új név már szerepel a tartalomjegyzékben), akkor nem kapunk hibajelzést, ill. hibaüzenetet, csak a lemezegység jelzi a hibát. Végül vesszővel vagy az ON kulcsszóval elválasztva megadható a lemezegység száma, amelyet U betű vezet be. Ennek az alapfeltételezés szerinti értéke 8.

Csak lemezen tárolt állományt lehet átnevezni.

RENUMBER renU P újraszámoz, átszámoz

Alapértelmezésben 10-es kezdősorszámmal és 10-es lépésközzel átsorszámozza a programot. Paraméterként megadható az új kezdősorszám, a lépésköz, amiknek alapértelmezése egyaránt 1Ø, továbbá a régi kezdősorszám, amelynek alapértelmezése a program első érvényes sorszáma.

Ha a régi kezdősorszámot (amely nem feltétlenül létező) úgy adjuk meg, hogy átsorszámozás után a sorszámok nem szigorúan növekvők lennének, akkor hibajelzést kapunk.

A parancs helyesen kezeli a programbeli sorszámhivatkozásokat (ELSE, GOSUB, GO TO, GOTO, RESTORE, RESUME, RUN, THEN, TRAP után).

Ha a végrehajtás során a sorszám túllépi a 65 535-öt, akkor a túlcsordulást a gép

levágja, és a sorszámot továbbra is 16 bites egészként kezeli. Erre különösen ügyeljünk, mert a sorszámhivatkozásokat ez az átírás megzavarhatja (pl. nem talál meg bizonyos sorokat). További gondot okozhat egy újabb átsorszámozás; ezután ugyanis előfordulhat, hogy egy immár kétszer szereplő sorszám miatt a hivatkozás nem az eredetileg megadott sorra fog mutatni.

Csak parancsüzemmódban használható.

RESTORE * reS U helyreállít, visszatesz

Alapértelmezésben a program első utasítására állítja a DATA-mutatót (innen kezdve keresi a következőnek beolvasandó adatot). Ha az utasításban sorszámot is megadunk (ennek léteznie kell a programban), akkor a megadott sor elejétől kezdi a keresést.

RESUME resU U újrakezd, folytat

A hibakezelő rutint záró utasítás. Alapértelmezésben a hibát okozó utasításra tér vissza. Paraméterként (érvényes) sorszámot megadva a sorszám által kijelölt utasításra, ill. a NEXT kulcsszót megadva, a hibát okozó utasítás utáni utasításra tér vissza.

Visszatérés után a hibaállapot megszűnik. A hibakezelő rutinok nem ágyazhatók egymásba. Természetesen csak programban fordulhat elő.

RETURN * reT U visszatér, visszajön; visszatérés

A szubrutint lezáró utasítás. Végrehajtásakor a GOSUB utáni utasításra tér vissza, törölve a visszatérési verem tetejét (és visszatöltve az aktuális sorszámot). Ha nem szubrutinban adjuk ki, akkor az üres GOSUB-verem miatt hibajelzést kapunk.

Nyilvánvaló, hogy csak programban fordulhat elő, bár szubrutin parancsból is hívható.

RGR rG NF Recall GRaphic: visszahívja a GRAPHIC értékét

Az álargumentumot használó függvény az aktuális grafikus üzemmódot (ennek kódját) adja. Az argumentumnak sem típusát, sem értékét nem vizsgálja, de a meglétét igen.

RIGHT$ * rI TF jobb oldal

A függvény argumentumként egy szöveget és egy számot vár, és a szövegnek a szám által előírt hosszúságú jobb oldali részét adja. Ha az argumentumban megadott szöveg az előírtnál rövidebb, akkor azt az eredeti hosszúsággal kapjuk vissza. A számargumentum csak pozitív, 256-nál kisebb lehet.

RLUM rL NF Recall LUMinousness: visszahívja a fényerősség értékét

Az argumentum által kijelölt képernyőtartomány (zóna) fényességét szolgáltató függvény. Az argumentum a COLOR-utasítás első, a függvényérték a harmadik paraméterének felel meg. Az argumentum egészrésze tehát legalább Ø és legfeljebb 4. (L. még **COLOR, RCLR**!)

RND * rN NF RANDOM: találomra, vaktában, véletlenszerűen

Az argumentumtól függő módszerrel 1-nél kisebb, nem negatív véletlen számot állít elő, és ennek megfelelő állapotba jut. Ha az argumentum Ø, akkor az addigi állapotoktól független, csak a belső óra aktuális értékétől függő véletlen számot állít elő. Negatív argumentum esetén csak az argumentumtól függő értéket ad. Pozitív argumentum esetén az argumentum értékétől független, csak a véletlenszám-generátor előző állapotától függő álvéletlen számot szolgáltat.

Az álvéletlen számok valóban véletlen kezdőértékét pl. a Ø vagy –TI argumentummal végrehajtott függvényhivatkozással állíthatjuk elő. Ha itt konstans negatív argumentumot adunk, akkor mindig ugyanazt a számsorozatot kapjuk.

RUN * rU P fut, jár, futtat; járat, futás

A tárolt programot indító parancs. A futás alapértelmezésben a program elejétől indul, ha sorszámot is megadunk, akkor a kijelölt (létező) sortól.

Programban is megadható, de végrehajtás után (a program végén) ismét parancsüzemmódba kerül a rendszer.

A végrehajtás a rendszer alapállapotba állításával kezdődik, majd a vezérlés a program elejére, ill. a megadott sorra adódik. A képernyő állapota nem változik, sem a grafikus képernyő definícióját, sem az állapotát nem törli.

Az f_6-billentyűvel is elérhető.

SAVE * sA U megment, félretesz

Az éppen tárolt programot kazettára vagy lemezre írja. Első paraméterként a létrehozandó állomány nevét kell megadni, amely csak a kazettás egységre való mentéskor hagyható el. A második paraméter az egység száma. A harmadik paraméternek csak kazettás egység esetén van jelentősége: az alapértelmezés szerinti Ø érték mellett úgy menti ki a programot, hogy az betöltéskor az aktuális BASIC-területre kerül. Ha a paraméterérték 1, akkor a betöltési cím is kimentődik. A paraméterrel előírhatjuk a szalagvégjel (EOT) rögzítését is, ha ennek értéke 2. Ha ez utóbbi kettőt együttesen akarjuk előírni, akkor a paraméter értékét 3-nak adjuk meg.

Programból megadva a végrehajtás után a futás tovább folyik. A program magát menti ki. (L. még **DSAVE**, monitorfunkciók.)

Ha a név előtt, az idézőjelben a kereskedelmi jelet @ és egy kettőspontot is megadunk, akkor törli az esetleg már létező ilyen nevű állományt.

SCALE scA U fokbeosztás, skála, lépték

A grafikus léptéket átértelmező utasítás.

Paraméterének Ø értéke mellett az alapértelmezés szerinti lesz a grafikus képernyő beosztása, míg 1 esetén a grafikus képernyő 1Ø24×1Ø24 méretű lesz. Ez utóbbi csak a logikai kezelést változtatja meg, a képernyő fizikai felbontása változatlan marad. Természetesen ilyenkor eltorzul a képernyő; az alakzatok függőleges irányban mintegy összenyomódnak (merőleges affin képet kapunk).

Ügyeljünk arra, hogy ezt az értelmezést pl. a RUN sem állítja alaphelyzetbe!

SCNCLR sC U SCreeN CLeaR: képernyőtörlés

Az aktuális képernyő(k) tartalmát törli. Az f_4-billentyűvel is elérhető.

Ügyeljünk arra, hogy osztott megjelenítés esetén a grafikus és a szöveges képernyő egyaránt törlődik!

SCRATCH scR U (meg)vakar, (ki)kapar, karcol

Az első paraméterben megadott nevű lemezes állományt törli a tartalomjegyzékből. (A név megadásában használhatók a jolly-karakterek is.) A másodikként megadható paraméter a meghajtó számát írja elő, amelyet a vessző után D betű vezet be. Harmadik paraméterként a lemezegység számát adhatjuk meg, amit vessző vagy az ON kulcsszó után U betű vezet be. Ezen paraméterek alapfeltételezés szerinti értéke Ø, ill. 8.

Ha parancsüzemmódban használjuk, akkor a rendszer az „ARE YOU SURE?" (Biztos benne?) üzenet kinyomtatása után válaszra vár: ha erre egyetlen Y-t tartalmazó sorral válaszolunk, akkor végrehajtja a parancsot. Ha a válasz N betű, vagy érvénytelen, akkor a parancs végrehajtása elmarad. A végrehajtás után kiírja a parancs eredményét és a lemez státuszát.

SGN * sG NF signum: előjel

Az előjelfüggvény értéke 1, ha az argumentum pozitív, −1, ha negatív és Ø, ha az argumentum is Ø.

SIN * sI NF szinusz

Az ívmértékben értelmezett argumentum szinuszát adja.

SOUND sO U hangzik, hallatszik, hangot hallat

Az utasítás utáni három paraméter rendre a (hang)csatorna kijelölését, a hangmagasság megadását és az időtartam beállítását végzi. A hanggenerátor kétcsatornás: az első csatorna egy hang kiadására használható a paraméter 1-es értéke mellett, a második csatorna egy hang vagy zaj generálását teszi lehetővé, ha a paraméter értéke 2, ill. 3. Az első paraméternek ezek a megengedett értékei.

A második paraméter a hangmagasságot határozza meg, értéke 1Ø24-nél kisebb, de nem negatív. A Hz-ben megadott „F" frekvenciát úgy állíthatjuk elő, hogy a második paramétert 1Ø24—111840.45/F értékűnek választjuk.

A harmadik paraméter a hang időtartamát szabja meg. Értéke Ø-tól 65 535-ig változhat, és Ø.Ø2 másodperces lépésközű időtartam megadását teszi lehetővé. Ha Ø időtartamot írunk elő, akkor azonnal kikapcsolja az adott hangot az előző időzítéstől függetlenül. (Kikapcsoláskor érdemes a bekapcsolt hang magasságát előírni, különben a kikapcsolás jól érzékelhető hanghatást eredményez.)

A hang időtartama alatt a program tovább fut, de újabb SOUND-utasítás csak Ø időtartam előírásakor hajtódik végre azonnal. Ha a hang időtartama alatt a következő nem Ø időtartamú SOUND-utasításra fut a program, akkor ez csak a már megszólaltatott hang befejeződése után hajtódik végre.

SPC(* sP F SPaCe: tér(köz), betűköz

A nyomtatási lista elemeként megadható függvény az argumentumban megadott számú jobbra léptetést hajt végre. (Nem szóközt ír!) Az argumentum 256-nál kisebb, nem negatív szám lehet.

SQR * sQ NF SQuare-Root: négyzetgyök

A nem negatív argumentum négyzetgyökét szolgáltató függvény. Az argumentum és a függvényérték egyaránt valós típusú.

SSHAPE sS U String SHAPE: szöveges alak, szöveges téglaforma

A grafikus képernyő egy téglalap alakú területét szövegváltozóba tölti. A kulcsszót egy változónév követi, amely kötelezően szövegváltozó; ebben tárolódik a grafikus terület tartalma. Ezt a tárolandó terület egyik csúcsának a koordinátái követik. Ezután megadható az átellenes csúcs két koordinátája. Ha ez elmarad, akkor ezt a csúcsot a grafikus kurzor jelöli ki. (A téglalapot tehát egy átlójának végpontjaival adjuk meg.) Az eredmény független a kijelölt átlótól, és a végpontok (csúcsok) megadásának sorrendjétől is.

A koordináták csak 65 536-nál kisebbek lehetnek, de a képernyőn kívül eső területet mindig töröltként értelmezi a rendszer.

Az utasításban negatív koordináták is állhatnak, de a rendszer ezeket is pozitívnak értelmezi (ráadásul kissé nehezen követhető módon). A negatív koordináták vizsgálatához a mintaprogramok között talál segítséget az Olvasó.

STEP * stE K lép, lépés, lépcsőfok

A FOR—NEXT-ciklus lépésközét bevezető kulcsszó: az utána megadott aritmetikai kifejezés aktuális értéke lesz a ciklus lépésköze. A lépésköz megadása elhagyható; alapfeltételezés szerinti értéke 1.

STOP * sT U megállít, véget vet

Végrehajtáskor „BREAK IN sorszám" üzenettel a futás leáll. Folytatni a CONT-paranccsal lehet. Az utasítás parancsban is megadható, ilyenkor a végrehajtás „BREAK" üzenettel áll le, és — mivel a parancs az I/O pufferben volt, ahol azt az újabb paranccsal felülírhattuk -— nem folytatható.

Elsősorban a program „belövéséhez" használatos; töréspontot hozhatunk vele létre.

STR$ * stR NSF STRing: (karakter)sor(ozat), zsineg, zsinór

Az argumentumként megadott szám nyomtatási képét adja. A szám pozitív előjelét szóköz helyettesíti, a szöveg vége az utolsó számjegy.

SYS * sY U SYStem: rendszer

Gépi kódú program BASIC-ből való indítását végző utasítás. A paraméter által megadott címen kezdődő gépi kódú programrészre adja a vezérlést. A paraméter 65 536-nál kisebb, nem negatív szám lehet.

A visszatérés az aktivizált program(rész) lezárásától függ: ha az a gépi kódú RTS-utasítással végződik, akkor végrehajtás után a rendszer visszatér a BASIC-be.

TAB(* tA F tabulátor

A nyomtatási lista elemeként megadható függvény. A kurzort az aktuális sor elejétől számítva az argumentumban meghatározott pozícióba teszi, ha még nem érte el ezt a helyet, ha elérte, akkor hatástalan. Az argumentum csak 256-nál kisebb, nem negatív szám lehet.

Ügyeljünk arra, hogy a TAB függvényt a nyomtatók esetenként másként értelmezik!

TAN* — NF tangens

Az ívmértékben értelmezett argumentum tangensét adja. (A függvény szakadásainak környezetében a számábrázolás korlátozott pontossága miatt a hivatkozás eredménye bizonytalan.)

THEN* tH K akkor, azután

Az IF-utasítás igaz ágának kulcsszava, a feltétel után áll. A vezérlés a THEN utáni utasításra adódik, ha a feltétel igaz (vagy nem Ø), különben az ELSE utáni utasításra, ennek hiányában a következő sorra kerül a vezérlés.

A THEN után közvetlenül sorszám is állhat, azaz a GOTO kulcsszó feltételes elágazáskor elhagyható. (L. még **GOTO**!)

Az igaz ág az ELSE kulcsszóig, ennek hiányában a sor végéig tart.

Az igaz ág üres is lehet, de a THEN kulcsszót kötelező kiírni. (L. még **ELSE, GOTO, IF**!)

TO* — K -hoz, -ig, -be, irányban, -nek, -ra

1. A FOR—NEXT-ciklus végértékének megadását bevezető kulcsszó. (L. még **FOR**!)
2. A GO-utasítás után megadva feltétel nélküli vezérlésátadást ír elő. (L. még **GOTO**!)
3. A DRAW-utasításban a végpont(ok) megadását bevezető kulcsszó.
4. A COPY-utasítás célkijelölésének kulcsszava.
5. A RENAME-utasítás új állománynevét bevezető kulcsszó.

TRAP tR U csapda, kelepce

Hibakezelő („elfogó") rutin szervezéséhez használható utasítás. Ha érvényes sorszámmal adjuk meg, akkor az utasítás végrehajtása után a hibaállapot hatására az előírt sorszámú sorra adódik a vezérlés, ahol a hibakezelő rutinnak kell lennie. A hiba kódját az ER, sorát az EL változók tartalmazzák. A hibajelzőt csak a hibakezelő rutint lezáró RESUME törli, így nyilvánvaló, hogy hibakezelő rutinok nem ágyazhatók egymásba.

Ha az utasítás nem tartalmaz paramétert, akkor a leírt funkció kikapcsolódik, azaz a rendszer hibakezelése lép (ismét) érvénybe, tehát hiba esetén a futás hibajelzéssel leáll.

Az érvénytelen sorra való hivatkozás programból nem kezelhető le.

(Ügyeljünk arra, hogy a hibakezeléssel felügyelt programrészben ne fordulhasson elő a rutin által le nem kezelt hiba, ill. arra, hogy a hibakezelő rutinban ne jöhessen létre hibaállapot. Ellenkező esetben nagyon áttekinthetetlen helyzet jön létre. Ebből adódik, hogy a program belövése előtt nem célszerű a programban TRAP-utasításokat elhelyezni, másrészt a hibakezelő rutinokat célszerű először a programtól elkülönítve kipróbálni, tesztelni, és csak ezután érdemes „élesben" is kipróbálni.)

TROFF troF U TRace OFF: nyom, pálya kikapcsolás

A nyomkövetési funkciót kapcsolja ki akár parancsból, akár programból. (L. még **TRON**!)

TRON trO U TRace ON: nyom, pálya bekapcsolás

Bekapcsolja a nyomkövetést, amelynek hatására minden utasítás végrehajtása előtt szögletes zárójelben kiírja az aktuális sorszámot. Programban és parancsként is megadható. Segítségével a program belövése végezhető egyszerűbben.

A nyomkövetést a program futása közben be-, ill. kikapcsoltatva a program egyes részeinek nyomkövetését is elérhetjük.

UNTIL uN K addig a pontig

A DO—LOOP-ciklus egyik fajta feltételének megadását bevezető kulcsszó. A ciklus addig folytatódik, amíg a feltétel igazzá nem válik. (L. még **DO, LOOP**!)

USING usI K használat

A nyomtatási formátumot vezérlő kulcsszó, amelyet a formátumot leíró szövegkifejezés követ. Ezután kötelezően pontosvessző áll, majd a nyomtatandó listaelemek, egymástól vesszővel elválasztva. A formátumvezérlés csak nyomtatási lista első elemeként (közvetlenül a PRINT után) adható meg, és a lista nem tartalmazhatja a TAB és az SPC függvényeket.

A formátumszöveg karaktereinek jelentése:

1. kettőskereszt — számjegy vagy karakterpozíció;
2. pluszjel — kiírandó előjel (helye);
3. mínuszjel — a negatív előjel (helye);
4. pont — a tizedespont helye;
5. vessző — a számot tagoló jel (helye);
6. dollárjel — a dollárjel helye;
7. négy felfelé nyíl — az exponens helye;
8. egyenlőségjel — a szöveg központosítását írja elő;
9. nagyobbjel — jobbra illeszti a szöveget.

1. A kettőskeresztek a számjegyek, ill. a szöveg karaktereinek helyét írják elő. Számuk meghatározza a számábrázolás pontosságát illetve tartományát, szöveg esetén annak hosszát.
2. A pluszjel a formátumszöveg elején vagy a végén állhat, ilyenkor a pozitív előjel is kinyomtatódik a megadott helyen. Dollárjel nem követheti.
3. A mínuszjel a formátumszöveg elején vagy a végén állhat, hatására az adott helyre kerül a negatív előjel, a pozitív előjel azonban nem íródik ki. Dollárjel nem követheti.
4. A tizedespont formátumszövegen belüli helye az egészrész és a törtrész jegyeinek számát határozza meg. A formátumszövegben csak egy tizedespont lehet.
5. A vesszővel a sokjegyű számokat tagolhatjuk. A formátumszövegben a kettőskeresztek között állhat, akár több helyen is.
6. A dollárjel a formátumszöveg elején, az előjel előtt (ha van), ill. a végén állhat, ha ott nincs előjel.
7. Az exponens helyét négy egymás melletti exponensnyíl jelöli ki. Az exponens kötelezően négy karaktert tartalmaz.
8. Az egyenlőségjel hatására a szöveg a formátum által kijelölt mező közepére kerül úgy, hogy a jobb oldalon üresen maradó karakterhelyek száma nem nagyobb a bal oldalon üresen hagyott karakterhelyek számánál. Ha a szöveg túl hosszú, akkor balra illeszt, jobb oldalon levág. A formátumszövegben tet-

szőleges számú egyenlőségjel lehet, de legalább egy kettőskeresztet is tartalmaznia kell.
Számot úgy nyomtat, mintha csak kettőskeresztből állna a formátumszöveg.

9. A nagyobbjel hatására jobbra illeszti a szöveget, ha az nem túl hosszú. Minden egyéb esetben hatása az egyenlőségjelével azonos.

Ha a formátumszöveg csak kettőskeresztet tartalmaz, akkor balra illeszt, túl hosszú szöveg esetén jobbról levág.

Ha a formátumszöveg nem tartalmaz előjelmegadást, akkor csak a negatív előjel íródik ki közvetlenül a legnagyobb, de legalább az egyes helyiértékű számjegy elé.

Ha egy szám az előírt formátumban nem nyomtatható ki (kerekítve sem), akkor nyomtatáskor a formátummezőben mindenütt csillag jelenik meg.

A formázott nyomtatás nagyon sok lehetőséget kínál, amelyet a PUDEF-utasítás tovább gazdagít.

USR * uS NF USeR: használó

A függvény az argumentumot a lebegőpontos (szoftver-) akkumulátorba tölti, majd az 1281, 1282 címeken tárolt címre adja a vezérlést. Ezzel egy gépi kódú programot indít, amiből visszatérve a függvény értéke a lebegőpontos akkumulátor tartalma lesz.

VAL * vA SNF VALue: érték, értékel

Az argumentumként megadott szöveg számértékét adja.

A függvény értéke Ø, ha a szöveg nem előjellel, számjeggyel vagy ponttal kezdődik. A szöveget balról jobbra konvertálja (értelmezi) mindaddig, amíg annak értelme van.

A függvény kezeli a karakterisztikát, de a műveleteket nem. Túl nagy abszolútértékű szám esetén túlcsordulási hiba keletkezik.

VERIFY * vE U igazol, bizonyít, átvizsgál

A kimentett programot ellenőrző utasítás. Paraméterei megegyeznek a LOAD-utasításéval, a végrehajtáskor azonban nem tölt be, hanem a kimentett programot visszaolvasva összehasonlítja az aktuális programmal.

Programból végrehajtva az ellenőrzés után a futás folytatódik, parancsüzemmódban „OK" jelzést kapunk. Ha az ellenőrzés eredménye negatív, akkor hibajelzést ad a rendszer.

Nincs lemezre vonatkozó változat, mint pl. **LOAD** és **DLOAD.**

VOL vO U VOLume of sound: hangerő

A hanggenerátor hangerejét szoftver úton vezérlő utasítás. A paraméter 9-nél kisebb nem negatív szám lehet, aminek egészrészét használja az értelmező. Ø-ra lekapcsol, 8-ra maximális hangerőt állít be.

WAIT * wA U vár(akozik)

Az utasítás kettő vagy három paramétert vár.

Az első egy tárcímet jelent, értéke tehát nem negatív és 65 536-nál kisebb, a másik kettő 256-nál kisebb és szintén nem negatív.

Végrehajtáskor először a megadott tárcím és a harmadik paraméter értékével

kizáró *vagy* műveletet, majd ennek eredményével és a második paraméter értékével logikai *és* műveletet végez. Ha ennek eredménye nem Ø, akkor a végrehajtás folytatódik, ellenkező esetben megismétli a fentieket. Ha a harmadik paramétert elhagyjuk, akkor az utasítás úgy értelmeződik, mintha Ø-t adtunk volna meg, azaz a kizáró *vagy* művelet eredménye éppen a tárcím tartalma lesz.

Figyelem! Az utasítás hatására a rendszer könnyen végtelen ciklusba kerülhet, amiből csak a RESET gomb segítségével szabadíthatjuk ki.

WHILE wH K mialatt

A DO—LOOP-ciklus egyik fajta feltételének megadását bevezető kulcsszó. A ciklus mindaddig folytatódik, amíg a feltétel igaz értékű. (L. még **DO, LOOP**!)

8. A C16 MONITORJA

„Nem lenne semmi baj a haladással, csak nem akar vége lenni."

Ogden Nash törvénye

A C16 számítógép rendelkezik egy beépített monitorprogrammal. Ez egy önálló működési egység, amely a gépi kódú programozást segíti. Természetesen ez nem jelenti azt, hogy a gépi kódú programozáshoz egy professzionális eszköz áll rendelkezésünkre. Elsősorban kisebb gépi kódú programok vagy még inkább rutinok írására használható, és alkalmas a gépi kódú programozás alapjainak a bemutatására.

Minthogy a monitor nem tartozik a BASIC eszköztárához, de haladó szinten sok esetben segítséget jelenthet, így célszerűnek látszik a monitorfunkciók áttekintő ismertetése.

A monitor minden számot hexadecimális alakban vár, így ezt nem kell külön jelezni. A címek legfeljebb négyjegyűek lehetnek, a byte tartalma legfeljebb kétjegyű. A paramétereket szóközök választják el egymástól.

Monitor üzemmódban is a teljes képernyő használható szerkesztésre, természetesen minden funkcióhoz.

A monitor bejelentkezéskor kiírja a regiszterek tartalmát (módosításhoz közvetlenül használható alakban), és a kurzort a következő sor elejére állítja. Ilyenkor parancsra vár.

A TEDMON parancsai a következők.

A (Assemble), ponttal is helyettesíthető

Az utána álló címre tölti a cím után megadott assembly nyelvű utasításnak megfelelő kódo(ka)t, majd a soron következő utasítás címét írja ki egy újabb A betű után a következő sorba. Így a fordítás tovább folytatható. Ha erre üres sort adunk válaszként, akkor a fordítást befejezi.

C (Compare)

A parancs három címet vár; az első kettő egy tártartományt határoz meg, ennek tartalmát hasonlítja össze a harmadik címnél kezdődő (ugyanekkora) tárterület tartalmával. Az eltérő byte-ok címeit az első két paraméter által meghatározott címtartományban adja meg. Az öszehasonlítás növekvő címek irányában halad, így az FFFF címet a 0000 cím követi. A végcímre már nem hajtódik végre.

D (Disassemble)

Az aktuális vagy a megadott kezdőcímtől kezdve a tárban levő kódokból assembly nyelvű listát készít úgy, hogy a végcímet, ennek hiányában a kezdőcímtől számított 20. byte-ot még értelmezi. A sor elejére kiírt pont felhasználható a szerkesztés utáni befordíttatáshoz.

F (Fill)

A parancs három paramétert vár. Az első két paraméter által meghatározott címtartományt (a végcímet már nem!) feltölti a harmadik paraméter által megadott értékkel.

G (Go)

Az aktuális címtől (regisztertartalmat módosítva beállíthatjuk a PC értékét), vagy a megadott címtől elindítja a gépi kódú programot. Ha ebből a TEDMON monitorba kívánunk visszatérni, akkor erre a BRK-utasítást használjuk a programban.

H (Hunt)

A parancs legalább három adatot vár. Az első kettő a keresés címtartományát írja elő, a további(ak) a keresett adatcsoportot. Ez utóbbit megadhatjuk byte-onként vagy aposztróf után karaktersorozattal. A keresett adatcsoport első eleme mind a kezdőcím tartalmával, mind a végcím tartalmával összehasonlítódik.

Az aposztróf előtt is kötelező a szóköz, de az utána álló már a szöveg részeként értelmeződik, míg a karaktersorozatot záró szóközt figyelmen kívül hagyja. Nincs mód byte-ok és szövegek vegyes megadására, így szóközzel záródó karaktersorozatot csak byte-onként adhatunk meg.

L (Load)

Az idézőjelek között megadott nevű programot tölti be az adott számú egységről. Az egység kazettás vagy lemezes egység lehet. Az adatokat vessző választja el.

M (Memory)

A parancs legfeljebb két címet vár. Az általuk meghatározott tartomány tartalmát kiírja a következő alakban: nagyobbjel, majd az elsőnek kiírt byte címe, ezután szóközökkel elválasztva nyolc byte tartalma hexadecimális alakban, majd kettőspont után inverz ASCII megjelenítéssel (a 7. bitet elhagyva) a byte-ok tartalma. Ha végcímet nem adunk, akkor 12 sor íródik ki, ha a kezdőcím elmarad, akkor az aktuális címtől kezdve listáz, ilyenkor a listázás folytatólagos.

Ha egy címet adunk meg, akkor az kezdőcímként értelmeződik. A határokban megadott címek tartalma mindig kiíródik. A nagyobbjel felhasználható a tártartalmak módosítására.

R (Registers)

A parancs hatására kiíródik a processzor regisztereinek tartalma. Egy sorban a regiszterek nevei, alatta pontosvessző után a regisztertartalmak állnak. A pontosvessző lehetővé teszi a regiszterek tartalmának indítás előtti beállítását.

S (Save)

Az idézőjelek között megadott néven kimenti a megadott számú egységre a kezdőcím és a végcím paraméterekkel megadott tártartományt. A végcím már nem tárolódik. A létrehozott állományt programként kezeli, amely az első két byte által meghatározott címre töltődik. A paramétereket vesszők választják el.

T (Transfer)

Az első és a második cím által kijelölt tárterületet áttölti a harmadik címtől kezdődő tartományba. A végcím tartalma is még áttöltődik.

V (Verify)

Ellenőrzi, hogy a megadott nevű program egyezik-e a tár tartalmával. A vessző után a tárolóegység számát kell megadni. Eltérés esetén hibajelzést kapunk.

X (eXit)

Visszatér BASIC-be. Ilyenkor célszerű egy CLR-utasítást kiadni, hogy a rendszer mutatói alapállapotba kerüljenek.

9. NÉHÁNY BASIC-TRÜKK

„A négykerékmeghajtás haszna csak annyi, hogy elhagyatottabb helyeken akad el az ember."

Phillips törvénye

A következőkben olyan BASIC-ben megvalósítható fogásokról lesz szó, amelyek a kódolásban hasznosíthatók a program különböző szempontok szerinti javításában. Ezek olyan apróságok, melyeket érdemes megtanulni a gép sajátságainak minél jobb kihasználása érdekében. A teljességre törekedni itt végképp értelmetlen, hiszen a trükkök szinte kimeríthetetlenek, csak elég mélyen kell ismerni hozzá a rendszert. Ebből adódik, hogy az alábbiak csak amolyan ötletforrásnak tekintendők, az Olvasón a sor, hogy továbbfejlessze ismereteit.

9.1. VALÓS—EGÉSZ SZÁMKONVERZIÓ

„A tapasztalat olyasvalami, amihez csak közvetlenül azután jut az ember, hogy szüksége lett volna rá."

Oliver törvénye

A C16 egészen sajátosan oldja meg a valós számok egésszé való átalakítását. Ha a valós szám nem esik az egész típusú számok ábrázolási tartományába, a konverzió akkor is végrehajtódik. Ilyenkor az történik, hogy a számból a „kettedespont" előtt álló 16 bit marad meg, amely előjeles egészként értelmeződik. Ez azért fontos, mert a számok ilyenkor a magasabb helyiértékű biteken (is) csonkulhatnak, ha viszont a szám nagyon nagy abszolútértékű, akkor a megfelelő 16 bit már kikerül az ábrázolási tartományból, így az eredmény Ø lesz. Hibajelzést — ellentétben a COMMODORE 64-gyel — a C16 ilyenkor sosem ad.

Gépünknek ezt a tulajdonságát jól használhatjuk pl. a grafikánál. Ha a vízszintes koordinátaértékeket X AND 511, a függőlegeseket Y AND 255 alakban írjuk elő, akkor egy 512×256-os méretű olyan „képernyőt" érzünk, amely vízszintesen is, függőlegesen is periodikus. Természetesen a képernyő valójában csak a bal felső 320×2ØØ-as területet jeleníti meg.

A valós—egész konverziónak a későbbiekben is látjuk még néhány alkalmazását.

9.2. A BINÁRIS ÉS A DECIMÁLIS ARITMETIKA

„A bölcs az igazság fölfedezésén örvendezik, a bolond a hamisságén."

Gabirol megfigyelése

Számítógépünk belső számábrázolása bináris, azaz minden bevitel és kiírás során át kell alakítani a számokat. Mivel a gép ezt az átalakítást — a bevitelhez és a kiíráshoz képest — igen gyorsan végzi, így ez különösebb hátrányt nem jelent. A megoldás fő előnye a gyors aritmetikai működés. Természetes azonban az is, hogy ennek egyéb hatásai is érezhetők.

A bináris számábrázolásban végtelen tört lesz néhány olyan tört is, amely a tízes számrendszerben véges. Ebből néha meglepő eredmények adódnak. Pl. a

```
FOR I = Ø TO 1 STEP Ø.1: PRINT I: NEXT
```

ciklus az 1-es végértéket nem írja ki, de a lépésközt Ø.2-nek választva a végérték is rendben kiíródik. A fenti jelenség a gépnek tehát nem hibája, hanem tulajdonsága. A decimális aritmetika erénye tehát az, hogy kerekítéskor ugyanúgy hibázik, ahogy már megszoktuk. (Ennek viszont az az ára, hogy ugyanolyan pontossághoz több byte szükséges, és a műveleti sebesség is kisebb.)

Az elmondottakból adódik, hogy — különösen törttel való számoláskor — nem szabad ciklusfeltételként egyenlőséget megadni. Ez ugyanis csaknem biztosan végtelen ciklushoz vezet. Pl. az

```
I = Ø:DO UNTIL I = 1: PRINT I,: I = I + Ø.1: LOOP
```

ciklus végtelen ciklus, ráadásul a túlcsordulás miatt sem fog megállni, mert a növekmény előbb-utóbb kisebb lesz a számábrázolás pontosságánál. Feltételben egyenlőséget csak akkor adjunk meg, ha az egész (típusú) számokra vagy szövegekre vonatkozik.

9.3. ARITMETIKA LOGIKAI OPERÁTOROKKAL

„Ha végképp nincs más megoldás, próbáld ki főnököd javaslatát!"

Strano törvénye

Az eddigiek alapján már nyilvánvaló, hogy a számokon is végrehajthatók logikai műveletek. Ezekhez adnak néhány ötletet a következő példák.

Sokszor van szükség egy jelző átkapcsolására. Ilyenkor ezzel két lehetséges állapotot különböztetünk meg, a jelző ezeket azonosítja. Ezt a funkciót pl. az alábbi kis programrészlettel oldhatjuk meg:

```
IF A = Ø THEN A = 1 : ELSE A = Ø,
```

azonban ez eléggé hosszú és lassú is. Elegánsabb és gyorsabb is az átkapcsolást az

```
A = 1 - A
```

értékadással végezni; természetesen A értéke előzőleg beállítandó. Még hatékonyabb megoldás, hogy a jelző két lehetséges értékét Ø-nak és −1-nek választjuk, és az átkapcsolást egyszerűen az

A = NOT A

értékadással végezzük. A változót természetesen egész típusúnak is választhatjuk.

Bizonyos oszthatósági feladatok is könnyen kezelhetők, a bináris számábrázolás miatt éppen olyanok, amelyek tízes számrendszerben kissé kellemetlenek. A kettő hatványaival való oszthatóságot egyszerűen az alsó néhány bit vizsgálatával dönthetjük el. Ha ugyanis 2^n-nel való oszthatóságot kell vizsgálni, akkor a szám alsó n bitjét kell megnézni, ezek pedig éppen a megfelelő maradékot szolgáltatják. Így pl. A 4-gyel való oszthatóságát az alábbi sorral írathatjuk ki:

IF A AND3 THEN PRINT"NEM OSZTHATÓ":ELSE PRINT"OSZTHATÓ".

Ha egy képernyőn megjelenített karaktert inverz módban akarunk megjeleníteni, akkor kódját 128-cal kell megnövelni, ha 128-nál kisebb; ha nem, akkor semmit sem kell tennünk, hiszen a megjelenítés eleve inverz. Pl. a

FOR I = 3Ø72 TO I +39:POKE I,128 OR PEEK (I) : NEXT

sor hatására a képernyő legfelső sora inverz üzemmódba kapcsolódik.

A logikai műveletekkel tehát valójában bit szinten kezelhetjük az egész számokat. Ez számos lehetőséget kínál.

9.4. IF NÉLKÜL IS LEHET...

> *„A szélsőségek találkoznak."*
>
> *Colridge törvénye*

Köztudott, hogy az IF-utasítás végrehajtása meglehetősen lassú, amelyet egy esetleges vezérlésátadás még tovább lassít. Éppen ezért érthető, hogy a programokban érdemes „takarékoskodni" a feltételes utasításokkal. Erre az egyik lehetőséget a ciklusszervező utasítások kínálják, amelyek elsősorban a vezérlésátadást válthatják ki.

A következőkben olyan példákat nézünk, amelyekben nincs vezérlésátadás, a feltételvizsgálat elkerülése azonban célszerű.

Elsőként legyen az a feladatunk, hogy a *C* változó értékét 1-re állítsuk, ha A>B, és Ø-ra különben. Ez pl. az

IF A>B THEN C = 1 : ELSE C = Ø

sorral tehető meg. Helyette alkalmazható a gyorsabb

C = − (A>B)

értékadás. Természetesen ez utóbbi megoldás még gyorsabb, ha C értéke −1 és Ø kell hogy legyen, ugyanis ilyenkor az ellentett képzés elmarad.

A megoldás használható két szám közül a nem kisebb meghatározására is, azon-

ban így a sok művelet — és zárójel — miatt az IF-utasítással gyorsabb lesz a kiválasztás, mint a

C = − B * (A<B) − A * (A> = B)

értékadással.

A következőkben a *C* változó értékét véletlenszerűen csökkentjük, ill. növeljük eggyel. Ennek szokásos módja:

A = RND (1) : IF A<.5 THEN C = C−1: ELSE C = C+1.

Ennek egy kevéssel gyorsabb, de IF nélküli módja:

C = C + 1 + 2 * (RND (1)< .5).

Itt érdemes megjegyezni, hogy az IF nélküli megoldásokkal csak azért „versenyképes" a feltételvizsgálat, mert a C16 megengedi az ELSE használatát. Ha ezzel nem élünk, és a hamis ágat külön sorba írjuk, akkor a feltételvizsgálat végrehajtása lényegesen lassabb lesz.

9.5. A FOR—NEXT-CIKLUS EZEKET IS TUDJA

„Valószínűleg annak van igaza, aki habozik."

Bogovich törvénye

Sorsoljunk N felelőnek két-két feladatot, összesen K feladat közül, és a sorszámokat tegyük az A(N,1) tömbbe! A feladat szokásos megoldását az alábbi kis programrészlet mutatja.

```
100 FOR I = 1 TO N
110 A = 1 + INT (K * RND (1) * : B = 1 + INT(K * RND(1))
120 IF A = B THEN 110
130 A (I,0) = A:A(I,1) = B
140 NEXT I
```

A programrészlet már néhány apró ötlettel is lényegesen gyorsítható: a 120-as sor az előző folytatásaként is írható, másrészt a 110-es sorban a véletlen értéket azonnal a tömbbe tölthetjük, így a 130-as sor elmarad, ilyenkor azonban A helyére A (I,0) és B helyére A (I,1) írandó.

Elegánsabb — GOTO nélküli — megoldás az alábbi:

```
FOR I = 1 TO N: A(I,0) = 1 + INT (K * RND(I)) :
A(I,1) = 1 + INT (K * RND(I)) : I = I +(A(I,0) = A(I,1)):NEXT.
```

Ennek végrehajtási ideje közel azonos az előzőkben kijavított változatéval. Ezt a megoldást elsősorban az áttekinthetőbb programszerkezet miatt érdemes alkalmazni, vagy akkor, ha a programrészlet előtt sok sor van, így a feltételes ugrás lassan hajtódik végre.

A FOR—NEXT-ciklust több helyen is lezárhatjuk. Az alábbi, parancsként is megadható sor a programterületen sorvégeket keres, és kiírja a lehetséges címeket. Azt nézi meg, hogy a tárban melyik címeken tárolódik Ø, ugyanis ezek lehetnek sorvégek. (Erről hamarosan részletesebben is lesz szó.)

```
FOR I = 4Ø97 TO 6ØØØ : IF PEEK (I) = Ø THEN PRINT I; : NEXT :
ELSE NEXT
```

Természetes, hogy programban a feltételvizsgálat két ága különböző sorba is kerülhet, ilyenkor azonban ügyelni kell arra, hogy a ciklus esetleg akármelyik NEXT-utasítással is befejeződhet, tehát gondoskodni kell arról, hogy mindkét esetben jól folytatódjék a program végrehajtása.

Gyakori feladat, hogy ki kell íratni egy tömb Ø-tól különböző elemeit. Erre néhány megoldás:

```
1ØØ FOR I = 1 TO N
11Ø IF A (J) <>Ø THEN PRINT A (I)
12Ø NEXT
```

vagy

```
1ØØ FOR I = 1 TO N
11Ø IF A (I) = Ø THEN NEXT:GOTO 13Ø
12Ø PRINT A(I) : NEXT
13Ø ...
```

Az ELSE használatával lényegesen áttekinthetőbb és tártakarékosabb:

```
1ØØ FOR I = 1 TO N : IF A(I) = Ø THEN NEXT:ELSE PRINT A (I) :
NEXT
```

9.6. KERÜLJÜK A GOTO HASZNÁLATÁT!

> *„Ami vitatható, azon nem érdemes vitatkozni."*
>
> *Fahnstock harmadik vitatörvénye*

Miután megismertünk néhány vezérlésátadási trükköt, felmerül a kérdés: Mi a helyzet a GOTO-val?

A számítástechnikai szakemberek bár — vérmérsékletük és aktuális kedélyállapotuk függvényében — némileg eltérően nyilatkoznak, egyetértenek abban, hogy a GOTO-utasítás szükséges rossz. Ezt a BASIC nyelvvel ismerkedők általában nem értik meg, és különböző példákat konstruálva „bebizonyítják" a GOTO elkerülhetetlenségét. Gondolatmenetük több szempontból is hibás. Szó sincs ugyanis arról, hogy a GOTO száműzendő lenne minden programból. Azt azonban be kell látni, hogy egy sok ugrást tartalmazó program működését nehéz követni, és még ennél is nehezebb ellenőrizni. Alapvetően más a helyzet a szubrutin és a ciklus vezérlésátadásaival. Ezek ugyanis — akár ciklusról, akár szubrutinról van szó — minden esetben egy programszerkezetet írnak le; a program egy funkcionális egységét határolják. Egy GOTO-utasítás alkalmazásakor sosem tudhatjuk, hogy az egyszerű (és

egyszeri!) ugrás, vagy egy blokk határa, és ciklust szervez, és ha ez utóbbi, akkor hol a blokk másik vége, hol tesztel a ciklus, és így tovább.

Szólni kell a BASIC GOTO-utasításának két további hátrányáról. Az egyik, hogy az értelmező minden végrehajtáskor számmá alakítja a sorszámhivatkozást, amely meglehetősen lassú. Hasonlóan lassú a sor keresése, különösen akkor, ha a program végén a sor elejére vagy néhány sorral előbbre ugratunk.

Érdemes kipróbálni, hogy a vezérlésátadás milyen tulajdonságú. Írjuk be a következő programocskát!

```
1 I = Ø : K = 1ØØ : L = 1 : T = TI
2 I = I + L : IF I < K THEN 2 : ELSE PRINT TJ-T
```

Lefuttatva a programot a belső óra mérési eredményét kapjuk, amelynek időegysége hatvanad másodperc. Ha a programot átsorszámoztatjuk úgy, hogy a kezdősorszám ötjegyű legyen, akkor a futási idő mintegy 10 %-kal nő. Ennek oka, hogy a sorszámhivatkozás átalakítása tovább tart. Írjunk be e két sor elé 18 újabb sort! (Elég, ha ezek egy-egy kettőspontot tartalmaznak.) A futási idő ismét növekszik, most azonban több mint 30 %-kal!

A kapott eredmény remélhetőleg meggyőzte a még kételkedőket.

9.7. HASZNÁLJUK A DO—LOOP-CIKLUST!

> *„Egy gyerek nem gyerek, két gyerek sok a jóból."*
>
> *Otoole axiómája*

Párbeszédes üzemmódú (játék és gyakoroltató) programok esetében jó, ha a gép valamilyen módon jelzi az üzemkészségét még akkor is, ha éppen a felhasználásra vár. Ezt pl. úgy lehet megoldani, hogy mialatt a gép a billentyű leütésére vár, a képernyő színét kapcsolgatjuk. Két kapcsolás között természetesen időzíteni kell, miközben a billentyűzet figyelése folyamatos. DO—LOOP-ciklusokkal ez igen elegánsan megoldható:

```
DO: I = Ø: COLOR Ø, 2+15 * RND(1): COLOR 4,2+15 * RND(1) :
DO WHILE I<1ØØ :I= I+1 :GET A$:LOOP WHILE A$="":LOOP
WHILE A$=""
```

A megadott sor csak a rövidítéseket is használva gépelhető be, de több sorba is írható. Működése röviden a következő. A belső ciklus időzít és figyeli a billentyűzetet. Ha leütünk egy billentyűt, akkor mindkét ciklust azonnal befejezi. A külső ciklusban először az időzítés kezdőértékét állítjuk be, majd a színek beállítása után az időzítés következik.

Természetesen sok más esetben is érdemes a DO—LOOP-ciklust használni. Ha pl. nem lehet, vagy nem érdemes előre tudni, hogy a ciklusnak hányszor kell lefutni, vagy ha a magból kell kiugrani.

Az alábbi kis programrészlet az X szám arkuszszinuszát adja a C változóban.

```
100 B = π/2:A = -B: IF ABS(X)>1 THEN PRINT"HIBA" : STOP
110 DO: C = (A + B)/2:(I)F B -A<1E-7 THEN EXIT:ELSE E=SIN(C):
IF E<X THEN A =C: ELSE B = C 130 LOOP
```

A működése nyilvánvaló, a szakaszt addig felezzük, amíg a hossza az előírtnál kisebb nem lesz. A függvényérték közelítése ennél jobb.

Természetesen más inverz függvény értékének előállítása hasonlóan történhet, azonban a határok beállítását és a helyettesítési érték vizsgálatát megfelelően kell végezni.

9.8. MI VAN A TÁRBAN?

> *„Ne higgy a csodákban — támaszkodj rájuk!"*
>
> *Finagle 6. szabálya*

A kérdésre igen nehéz volna kimerítő választ adni. Néhány érdekességet azonban érdemes megjegyezni.

Az alsó 2 Kbyte a rendszer által használt adatokat tárolja. Ezek közül érdemes a BASIC-program kezdőcímét ismerni, erre később szükségünk lesz: a 43. és 44. tárcímeken található.

A következő hat byte a változóterület adatait tartalmazza, természetesen ezeket is alsó byte, felső byte sorrendben. A 45, 46-os címeken a változóterület kezdetének, a következő két byte a tömbváltozók kezdetének a címét tartalmazza, majd a 49, 50-es címeken az első szabad byte címe van, ide kell a következő tömbváltozót bevezetni.

A fentiek alapján érthető, hogy a tömbváltozókat hogyan lehet törölni. A

```
POKE 49,PEEK (47) : POKE 50 ,PEEK (48)
```

sor hatására a tömbterület törlődik.

Vigyázzunk arra, hogy az egyszerű változók ilyen módon való törlése már nem ilyen egyszerű! Az egyszerű változókra vonatkozó ciklusbejegyzés miatt esetleg törölni kell a veremtárat is, ami BASIC-ből reménytelen, kivéve a CLR-utasítást. Az éppen erre való.

Jó tudni azt is, hogy a billentyűzetpuffer az 1319-es címen kezdődő 10 byte. Ennek aktuális hosszát a 239-es címen kezeli a rendszer; ha ennek tartalma 0, akkor a puffer — logikailag — törölt. Természetesen mindez azt jelenti, hogy minden GET- vagy INPUT-utasítás végrehajtása előtt a puffert törölni kellene, nehogy korábbi — valójában már érvénytelen — billentyűleütés hatása érvényesülhessen. Ez különösen a billentyűre való várakozás esetén zavaró. Ilyenkor a

```
POKE 239,0 : GETKEY V$
```

sort érdemes használni, ennek védelme már megfelelő.

A második 2 Kbyte a képernyőtár: 2048-tól 3047-ig a színeket, 3072-től 4071-ig

a karaktereket tárolja. Mindkét terület után egy-egy 24 byte-os üres terület van, a rendszer ezeket nem használja, de erre sajnos nincs garancia.

A 4Ø96-os címen egy Ø byte található; ez előzi meg a programterületet, ami 12 Kbyte méretű. A grafikus terület — ha grafikus üzemmódba kapcsoltuk — ennek felső 1Ø Kbyte-ja.

Mint arról már korábban szó volt, a BASIC-programot láncoltan tárolja a gép. Ez azt jelenti, hogy minden sor elején tárolja a következő sor helyét tartalmazó adatot. Ezt a C16-os gép úgy oldja meg, hogy a sor első két byte-ja a következő sor kezdőcímét tartalmazza, amelyet a szintén két byte-on tárolt sorszám követ. Ezután az adatbyte-ok állnak (legalább egy!), majd a sor végét jelző Ø tartalmú byte. Ezek alapján két sort úgy toldhatunk össze, hogy az első sort záró byte-ot és a következő sor 4 byte-nyi fejrészét pontosan 5 adattal felülírjuk. Az összefűzés menete a következő:

1. Válasszuk ki a sornak egy olyan részét, amely öt egymás utáni karaktert tartalmaz, és ezt nem számjegy követi! Az öt karakter között nem szerepelhet kulcsszó (beleértve a műveleteket is!), de pl. változónév vagy sorszám lehet.
2. Gépeljük be a sor első részét a kívánt sorszámmal, majd a második részt a következő sorszámmal.
3. Kerestessük meg a kívánt sorvéget a következő paranccsal!

   FORJ=41Ø1 TO 6ØØØ:IF PEEK(J)=Ø THEN PRINT J;:NEXT:ELSE NEXT

   Ha a szóban forgó sor a program elején van, akkor az első kiírt szám éppen a keresett cím lesz. (Érdemes 256-nál kisebb sorszámokat használni; ekkor a sorszám felső byte-ja Ø, és az egymáshoz közeli címek növelik a felismerés biztonságát.)
4. Írjuk át a megfelelő byte-okat a kívánt karakterkódokkal. Az

   I = ...:A$ =" ...":FORJ= 1 TO 5:POKE I+J,ASC(MID$ (A$,J,1)) : NEXT

   parancsban I az előző paranccsal megtalált byte elé mutat, A$ az átíró karaktereket tartalmazó, (legalább) 5 karakterből álló szöveg.
5. Sorszámozzuk át a programot! Ezzel a rendszer elvégzi az összeláncolást.

Az eddigiek alapján úgy látszik, hogy egy NEW-val törölt program is visszaállítható hasonló módszerrel. Ez azonban nem igaz. Ha ugyanis a korábban leírt ciklussal kerestetjük a sorvéget, akkor ezzel (a ciklusváltozó bejegyzése miatt) elrontjuk a program elejét. A legegyszerűbb ilyenkor a BASIC terület mutatóját Ø-tól különböző értékre állítani, majd egy átsorszámoztatással az összeláncolást elvégeztetni. A

POKE 4Ø97,17 : RENUMBER

sor végrehajtása után a program újra használható.

Lehetőség van arra is, hogy a visszaállított programot (pl. lemezre) kimentsük, majd visszatöltsük, amikor is a töltés után történik az összeláncolás. A

POKE 4Ø97,17 : SAVE "RENEW", 8

parancs után adjuk ki a

DLOAD "RENEW"

parancsot. A betöltés után a program teljes értékű lesz.

9.9. PROGRAMOK ÖSSZETOLDÁSA (MERGE)

„Valahányszor a jeles, agg tudós kijelenti, hogy valami lehetséges, majdnem biztos, hogy igazat szól. Valahányszor úgy nyilatkozik, hogy valami lehetetlen, nagyon valószínű, hogy téved."

Clarke első törvénye

Programfejlesztés során gyakran van szükség programok egybefűzésére. Ilyenkor legtöbbször az egymás után betöltött program összeszerkesztésére van szükség, pl. azért, mert a készülő programhoz egy olyan rutinra van szükség, amelyet egy korábbi programban már megírtunk. Ilyenkor a két programot (összetoldva) betöltjük, majd a felesleges sorokat kitöröljük és a megfelelő hivatkozásokat beírjuk.

Sajnos a programok összetoldása csak trükkel oldható meg. Ez azonban viszonylag egyszerű, és szinte kockázatmentes, hiszen mindkét program tárolható szalagon vagy lemezen. Az eljárás a következő:

1. A tárban lévő (első) program végére állítjuk a program kezdetét jelző mutatót. Ezt a PRINT PEEK (45), PEEK (46) adatai alapján végezhetjük. A 45-ös címen az alsó byte van, így ennek az értékét kell csökkenteni 2-vel. Ha ennek értéke Ø vagy 1, akkor csökkentve 254, ill. 255 lesz az új érték, ilyenkor a felső byte-ot is eggyel csökkenteni kell. Az így kapott értékeket a 43-as, 44-es címekre töltjük.
2. Betöltjük a következő programot.
3. Ha további programot kívánunk hozzáfűzni, akkor térjünk vissza az 1. pontra.
4. Állítsuk vissza a program kezdetének a mutatóját és sorszámozzuk át a programot!
   POKE 43,1 : POKE 44,16 : RENUMBER

A fenti eljárás után teljes értékű programot kapunk, amely korlátlanul használható, szerkeszthető. Mindössze arra kell ügyelni, hogy az egyesített részekben ne legyen két azonos sorszám, az ilyen ugyanis megzavarhatja a hivatkozásokat, és az átszámozás után már vissza sem állítható. Ez ellen úgy védekezhetünk, hogy az éppen betöltött programot azonnal átszámoztatjuk először egy RENUMBER, utána RENUMBER 1, majd RENUMBER 2, és így tovább. Ezzel a módszerrel tíz programot fűzhetünk össze egy menetben.

A nem említett mutatókat a rendszer természetesen helyesen kezeli.

9.10. A PROGRAM LASSÍTÁSÁNAK „MESTERFOGÁSAI"

„Semmi sem számít túlságosan, és nagyon kevés dolog számít egyáltalában."

Lord Balfour állítása

Az időzítő ciklus használatát mindenki nagyon hamar megismeri. Ez nagyon sok esetben elengedhetetlen, az ilyen ciklusok jól kézbentartható ideig futnak, használatuk tehát gyakorlatilag egyszerű.

Az elegáns, kifinomult megoldások nem így lassítják a futást. Ezek a program érdemi részében húzzák az időt. Az igazán „mesteri" megoldásokban a képernyő akár percekig(!!!) is mozdulatlanul mered a felhasználóra. Sajnálatos, hogy a legtekintélyesebb publikációk az ilyen gyöngyszemeket úgyszólván teljesen mellőzik, pedig elemzésük nagyon tanulságos lehetne. A következők tehát elsősorban hézagpótlónak tekintendők — szinte irodalmi előzmények nélkül.

1. Használjuk a GOTO-utasítást, lehetőleg GO TO alakban!

   Különösen hatékony a megoldás, ha ez a program végére kerül, és a sor elejére, vagy néhány sorral előbbre mutat.

   A feltételes vezérlésátadással szervezett ciklusok kitűnő alkalmat kínálnak a fenti elvek alkalmazására.
2. Írjunk minél több megjegyzést a programba!

   A megjegyzés már önmagában is elegáns lehetőséget kínál, de igazi haszna csak akkor érvényesül, ha önálló REM sorokat írunk, amelyek ráadásul jó hosszúak, és a program elején vagy a ciklusmagban vannak.

   A legpazarabb megoldások — természetesen több fogást egyesítve — olyan hasznos információkkal látják el a lista elemzőjét, amelyet csak 20–30 darab csillaggal lehet leírni.

   A megjegyzések szépséghibája, hogy néhány elvetemült ember hamis információkat ír a programba, vagy a — saját maga számára beépített — helyes kommentárokat elfelejti a program javításának megfelelően módosítani.
3. Írjunk minél több felesleges sort!

   Az üres REM vagy az egyetlen kettőspontot tartalmazó sor különösen a GOTO (pardon! GO TO) hatását fokozza, de nem elhanyagolható az az idő sem, amíg egy ilyen sort futás közben átlép. Ebben a kategóriában azonban mégiscsak a több (teljes sornyi) kettőspontot tartalmazó sorok jelentik az igazi eleganciát. És ha mindez ciklusmagban van!...
4. Ne becsüljük le a szóközöket!

   Szerény képességük miatt ugyan nem lehet átütő sikert remélni alkalmazásuktól, de az olvashatóság ürügyén rengeteget becsempészhetünk belőlük a programba, és hát sok kicsi sokra megy. (Jó tudni, hogy az értelmező nem igényel szóközöket az elválasztáshoz; a szöveget balról jobbra értelmezi. Így pl. a ?NORA parancs hatására 0-t ír ki, mert PRINT N OR A alakúnak értelmezi. Ügyelni kell azonban arra, hogy az ehhez látszatra nagyon hasonló ?DORA már szintaktikus hibát eredményez, mert PRINT DO... alakban kezdi értelmezni, amely természetesen hibás.)
5. A DATA-utasítást írjuk a program elejére!

   Ez a trükk csak a kettőspontok szaporításához fogható. A DATA-utasítás minden byte-ját végig kell nézni az értelmezőnek, hátha kettősponttal jelzett utasításvégen áll, de a sorvéget egyébként is fel kell ismerni. Az idézőjel a helyzetet tovább bonyolítja.

   Bizonyára többen rájöttek az eddigiek alapján arra, hogy a lehetőségeket még nem merítettük ki: kétszeres haszon, ha a 0 adatot kiírjuk, mert azt külön át kell lépni, beolvasáskor pedig konvertálni kell. (Hasonlóan jó megoldás, hogy az üres szöveget két egymást követő idézőjellel adjuk meg, esetleg előtte és utána egy-egy szóközzel, de csak a jobb olvashatóság miatt.)

   A rövid DATA sorok — ilyenkor persze több kell — a GO TO hatékony-

ságát fokozzák, de a ciklusmagban állók is nagyon hatékonyak. A bennük lévő szóközök hatása is ciklusban érvényesül igazán.

6. Használjunk sok állandót!

   Ez a lassításnak igazán jól átgondolt formája. Világos, hogy a konstansokat nem kerülhetjük el, így tehát szó sincs semmiféle trükkről! Igaz, hogy a ciklusban sokszor végrehajtandó számkonverzió helyettesíthető változókereséssel, ha a számot — egyszeri konverzióval — egy éppen használaton kívüli változóba töltjük, és csak ezután térünk a cilus végrehajtására, de ez a kérdés egyéb lehetőséget is érint.

   Jó tudni, hogy az egyetlen (tizedes)pontot tartalmazó szám Ø-nak értelmeződik, de ezt érdemesebb egy Ø karakterrel megadni, mert ennek konvertálása lassabb.

7. Használjunk egész típusú változókat!

   Igaz, hogy ezek számábrázolási tartománya kisebb, de az aritmetikai egység valós típusúvá alakítja ezeket és csak azután végzi el a műveletet. Ráadásul tártakarékossági szempontból közömbös a változó típusa.

8. Használjunk minél több és minél hosszabb nevű változót!

   Míg az előző két trükk COMMODORE-sajátság, addig az itt említett meglehetősen általános, így széles körben használható elv. Hatékonysága fokozható azzal, hogy a legtöbbször hivatkozottakat vezetjük be legkésőbb. (A hosszú változónevek hátránya, hogy könnyen összekeverhetők, mert az értelmezők legtöbbje csak az első két karaktert azonosítja, így az IRAT és az IRAS ugyanazt a változót jelenti.)

9. A szubrutinokat a program végére tegyük!

   Ez a lehetőség a GOTO-hoz hasonló előnyöket ígér. A leggyakrabban hívott szubrutinok legyenek leghátul, így a kereséskor sokszor kell végigmenni több sorszámon.

10. A felsoroltaknak van egy igen érdekes mellékhatása. Csaknem mindegyik megoldás többlet tárterületet igényel. Ez egy 12 Kbyte szabad tárterületű gép használatánál különösen figyelemre méltó. Néhány ügyesen megválasztott trükk bevetésével megoldhatatlanná tehetünk egész egyszerű feladatokat is: erősen lelassítjuk a futást, és teleírjuk — üres információval — a tárat. Ha ez utóbbi nehezen menne (amely képernyőszerkesztő mellett elképzelhetetlen, mert a csillagokat csak egyszer kell beírni, utána csak a sorszámokat kell módosítani), akkor kényelmes lehetőségként használhatjuk a grafikus üzemmódo(ka)t, ez(ek) 1Ø Kbyte-ot azonnal lefoglalnak.

Minden bizonnyal az Olvasót is lenyűgözte ez a páratlan gazdagság, és talán az összefüggések sokoldalúsága sem hagy senkit hidegen.

A felvetések természetesen még sok elemzést igényelnek. A cél ezek elindítása, hiszen a számítógépek iskolai alkalmazása egy nagyon fontos feladatot hozott magával. Egy — sok adatot kezelő — demonstrációs program nem futhat akármeddig; ha ugyanis az óra alatt nincs a futásnak meggyőző eredménye, akkor a gép használata nemcsak felesleges, de káros is. Így tehát az oktatóprogramok (különösen a demonstrációs célúak) esetében nagyon fontos, hogy a bemutatáshoz szükséges sebességgel, tehát elég gyorsan fussanak. Ehhez a fentiekkel nagymértékben hozzájárulhatunk.

10. FELADATMEGOLDÁSOK

„Csak az Úristen tud találomra válogatni."

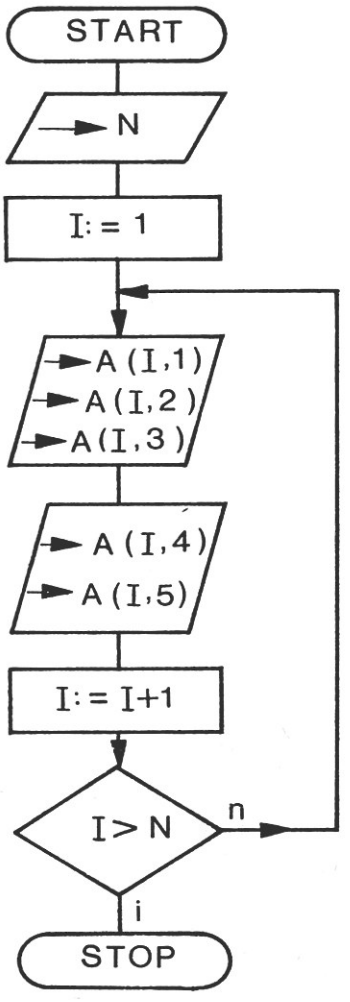
Levy kilencedik törvénye

Az osztálynyilvántartás például a 10. ábra szerinti folyamatábra alapján vihető be.

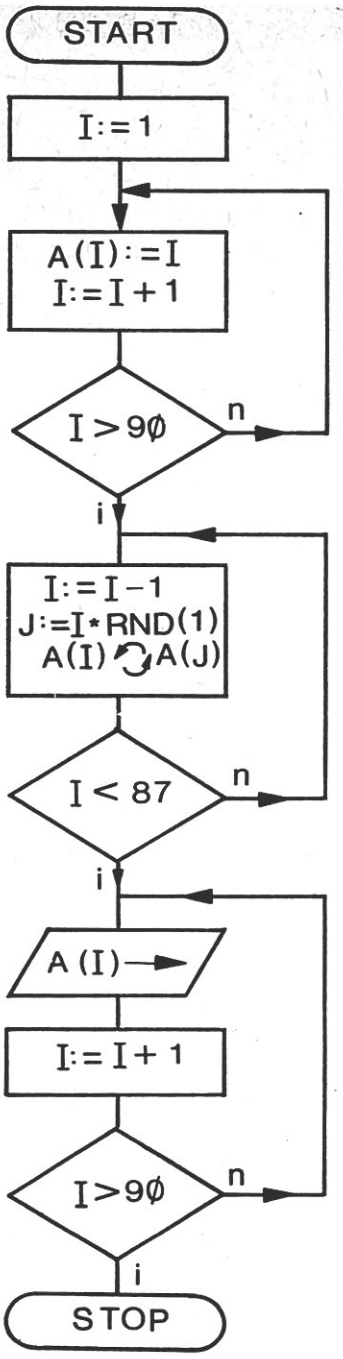
Lottószámok sorsolásának egyik lehetősége a 11. ábrán látható. A folyamatábrán szélsőségesen kihasználtuk a változók aktuális értékeit, ezeket a FOR—NEXT-ciklus működése miatt a kódoláskor fel kell adni, de pl. DO—LOOP-ciklust használva már érdekes a kódolás.

A kisorsolt lottószámokat úgy íratjuk ki növekvő sorrendben, hogy megkeressük a legkisebbet, kiírjuk, majd a helyére nem létező nagy számot írunk — ötször egymás után.

A 13. ábra a legkisebb elem keresésével történő rendezést mutatja. Érdemes



10. ábra

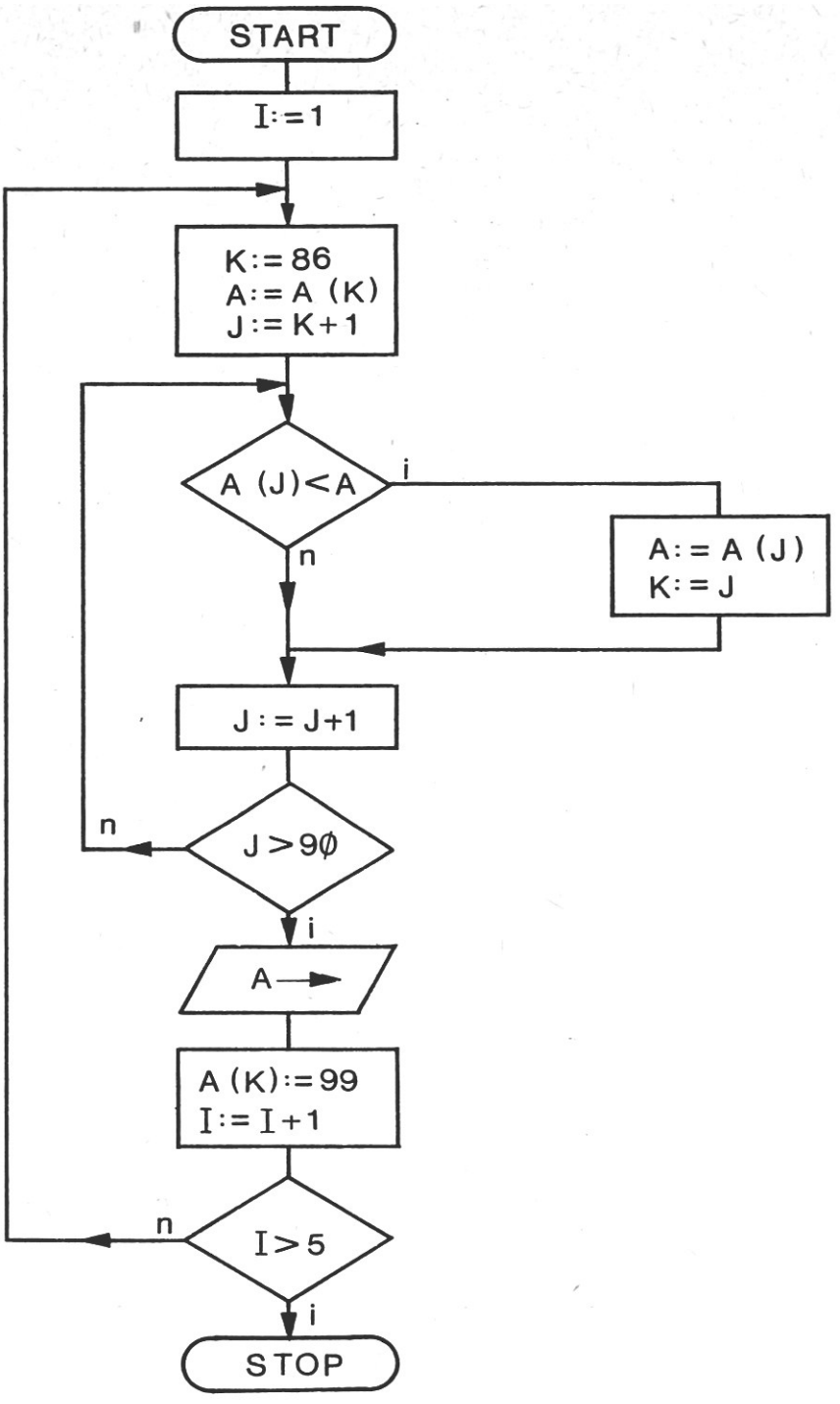


11. ábra

észrevenni, hogy a csere ebben az esetben két értékadással helyettesíthető. (Ezt index-tömb alkalmazásával lehet jól kihasználni, azonban ezzel ez a könyv nem foglalkozik.)

A rendezetten történő adatbevitel a 14. ábra alapján kissé bonyolultnak tűnik, de a megfelelő BASIC-program már kevésbé ijesztő. (Ezt a mintaprogramok között találja meg az Olvasó.) Csak nem túlságosan sok adat alkalmazásánál működik elég gyorsan, azonban a gyakorlatban ez a sebesség bőven elegendő (14. ábra).

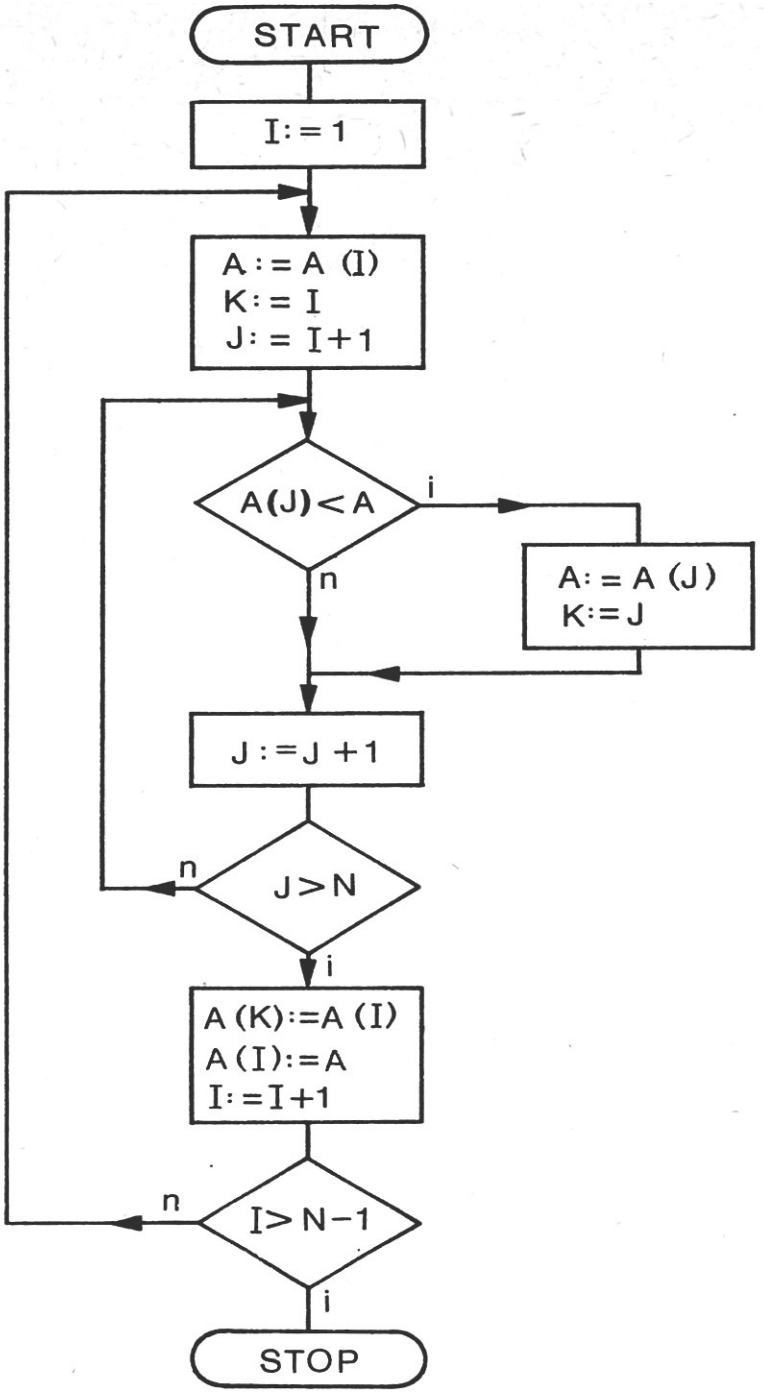
A lineáris kereséssel addig lépünk előre, amíg a keresett elem előtt vagyunk.



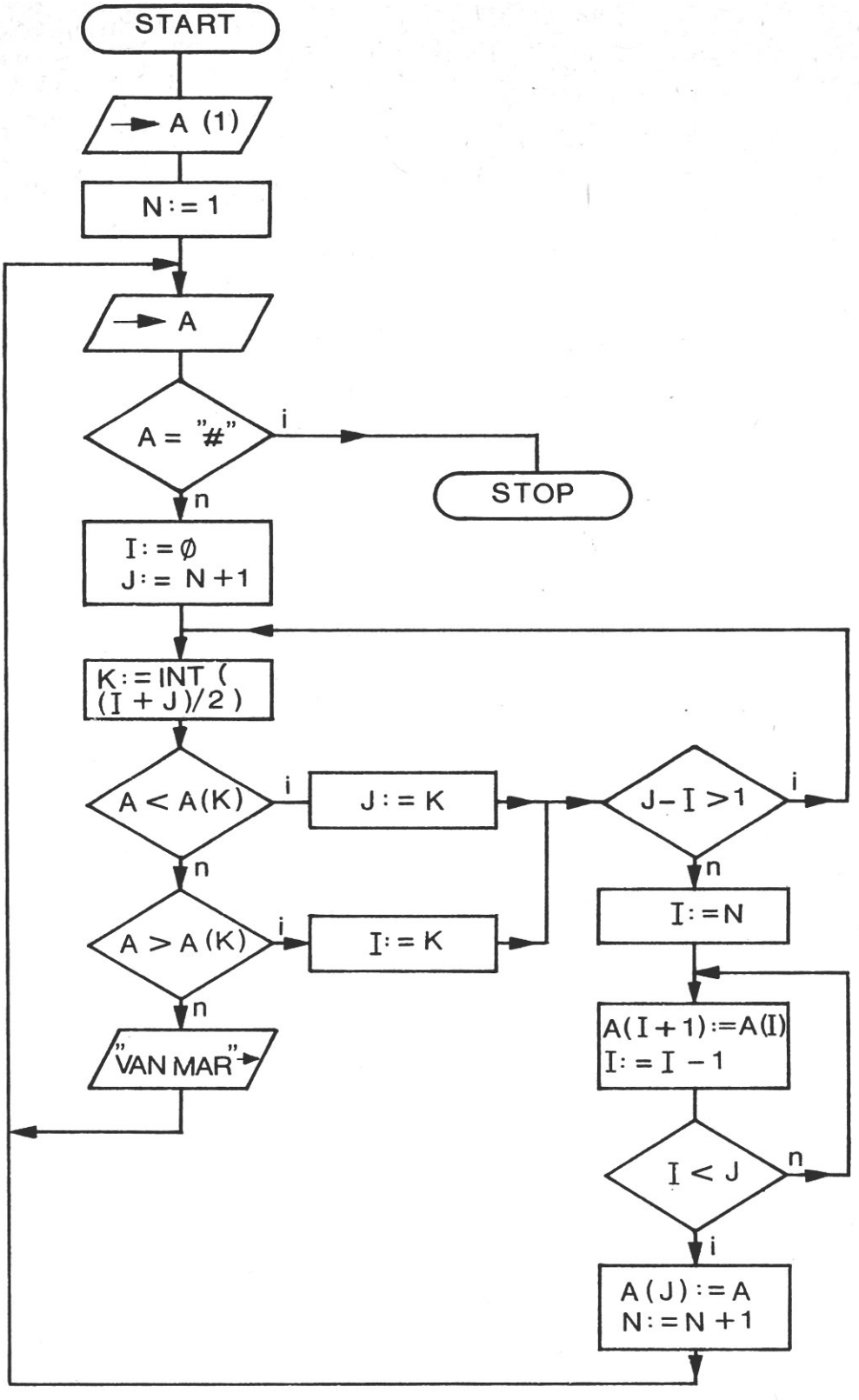
12. ábra

Ha így a tömb végére érünk, akkor nincs ilyen elem. Ha valamelyik lépés után már nem a keresett elem előtt leszünk, akkor megnézzük, utána vagyunk-e. Ha igen, akkor nincs ilyen elem; ha nem, akkor addig megyünk, amíg túl nem lépjük. Az ez előtti index az utolsó előfordulás indexe (15. ábra).

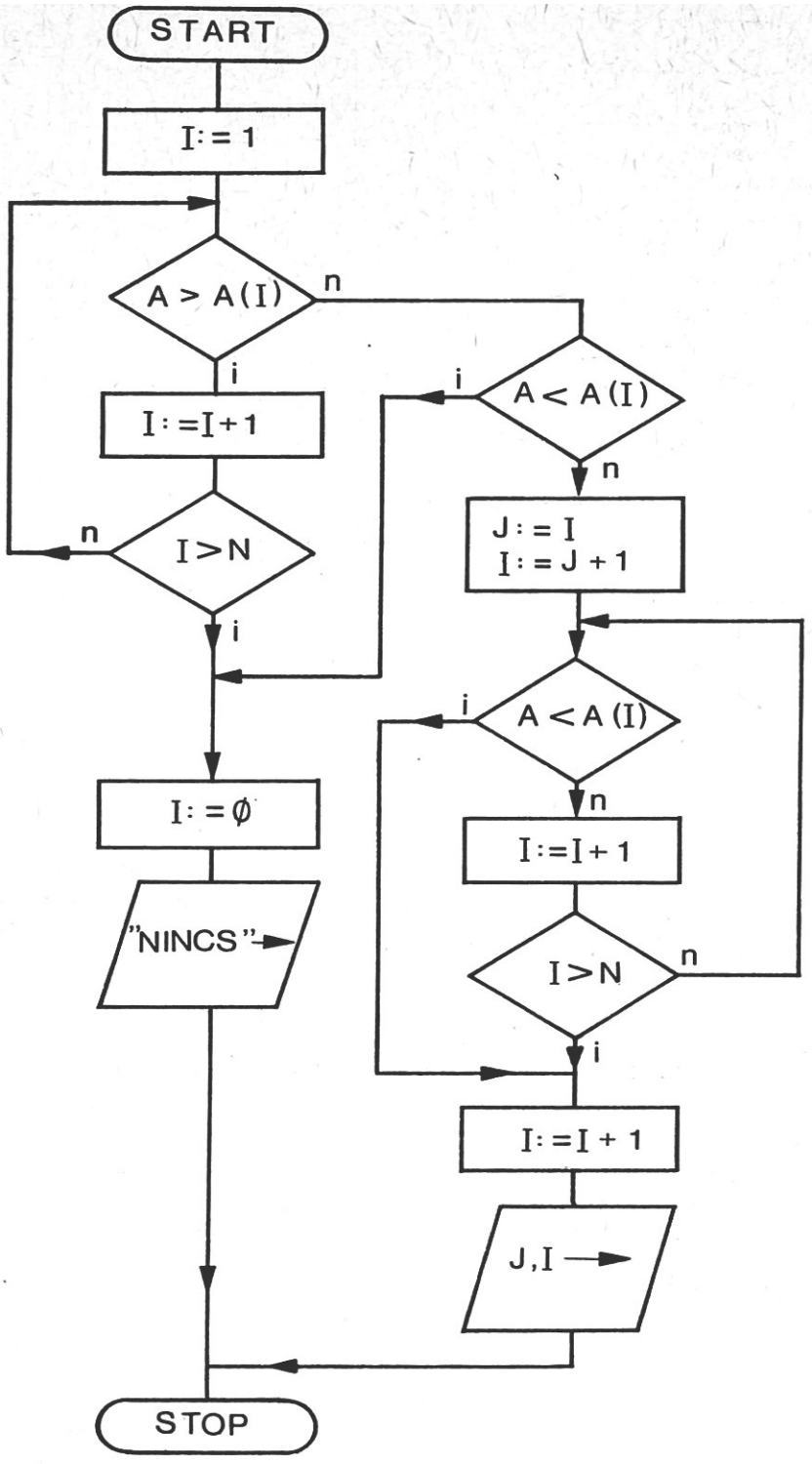
A három kódolási feladat megoldását (együtt) az 1. lista tartalmazza. Ez a megfelelő folyamatábrákkal együtt könnyen követhető. Ezek mintaprogramokba való beépítésével a későbbi programlistákon még találkozunk.


START
I:=1
A:=A (I)
K:=I
J:=I+1
A(J)<A
i
n
A:=A (J)
K:=J
J:=J+1
n
J>N
i
A (K):=A (I)
A (I):=A
I:=I+1
n
I>N-1
i
STOP

13. ábra


START
A (1)
N:=1
A
A = "#"
i
STOP
n
I:=∅
J:= N+1
K:=INT (
(I + J)/2)
A < A(K)
i
J:= K
J-I >1
i
n
n
I:=N
A > A(K)
i
I:= K
n
A(I+1):=A(I)
I:= I -1
"VAN MAR"
I < J
n
i
A(J):=A
N:=N+1

14. ábra


START
I: = 1
A > A(I)
n
i
I: =I+1
n
I>N
i
i
A < A(I)
n
J: = I
I: = J + 1
i
A < A(I)
I: = ∅
n
I:=I+1
"NINCS"
I>N
n
i
I: =I + 1
J,I
STOP

15. ábra

```
0 REM 1. LISTA
70 :
80 REM LEGNAGYOBB KOZOS OSZTO
90 :
100 IFV=0THENV=U:GOTO130
110 IFU<VTHENV=V-U:ELSEU=U-V
120 IFU>0THEN110
130 REM VEGE
140 :
170 :
180 REM MINIMUM
190 :
200 J=1:MI=A(J)
210 FORI=2TON:IFA(I)<MITHENMI=A(I):J=I:NEXT:ELSE:NEXT
240 :
250 :
260 REM RENDEZESEK
270 :
280 REM A SZOMSZED ELEMEK CSEREJEVEL
290 :
300 K=0:FORI=1TON-1:IFA(I)>A(I+1)THENA=A(I):A(I)=A(I+1):A(I+1)=A:K=-1:NEXT:ELSEN
EXT
310 IFKTHEN300
370 :
380 REM BUBOREKOS
390 :
400 FORI=1TON-1
410 FORJ=I+1TON:IFA(I)>A(J)THENA=A(I):A(I)=A(J):A(J)=A:NEXT:ELSENEXT
420 NEXT
470 :
480 REM A LEGKISEBB ELEM KERESESEVEL
490 :
500 FORI=1TON-1:A=A(I):K=I
510 FORJ=I+1TON:IFA(J)<ATHENA=A(J):K=J:NEXT:ELSENEXT
520 A(K)=A(I):A(I)=A:NEXT
```

11. MINTAPROGRAMOK

„Az életet csak visszafelé lehet megérteni, de csak előrefelé lehet leélni."

Kierkegaard megfigyelése

TÖMBÖK

A 2. listán a megoldások között szereplő két lottós folyamatábra BASIC-programja látható. Három további eljárás a 3. listán található. Ezek az előzmények alapján megérthetők; elemzésük sok tapasztalatot hozhat még a haladóbbak számára is.

```
0 REM 2. LISTA
10 :
20 REM LOTTO
30 :
40 CLR:DIMA(90):FORI=1TO90:A(I)=I:NEXT
50 FORI=90TO86STEP-1:J=I*RND(I)+1:A=A(I):A(I)=A(J):A(J)=A:NEXT
60 PRINT" A LOTTOSZAMOK:";:FORI=86TO90:PRINTA(I);:NEXT:PRINT
70 :
80 END
90 :
100 REM NOVEKVO KIIRASSAL
110 :
120 CLR:DIMA(90):FORI=1TO90:A(I)=I:NEXT
130 FORI=90TO86STEP-1:J=I*RND(I)+1:A=A(I):A(I)=A(J):A(J)=A:NEXT
140 PRINT" A LOTTOSZAMOK:";:FORI=1TO5:K=86:A=A(K)
150 FORJ=87TO90:IFA(J)<ATHENA=A(J):K=J:NEXT:ELSENEXT
160 PRINTA;:A(K)=99:NEXT:PRINT
0 REM 3. LISTA
70 :
80 REM LOTTO-1.
90 :
100 CLR:DIM A(5):A(1)=INT(1+90*RND(0))
110 PRINT" LOTTOSZAMOK:";A(1);:FORI=2TO5
120 A(I)=INT(1+90*RND(I))
130 FORJ=1TOI-1:IFA(I)=A(J)THEN120
140 NEXT:PRINTA(I);:NEXT:PRINT
170 :
180 REM LOTTO-2
190 :
200 CLR:DIMA(5):FORI=1TO5:A(I)=0:NEXT
210 J=INT(1+90*RND(0))
220 FORI=1TO5:IFA(I)=JTHEN210
230 IFA(I)=0THENA(I)=J:GOTO 250
240 NEXT
250 IFA(5)=0THEN210
260 PRINT" LOTTOSZAMOK:";:FORI=1TO5:PRINTA(I);:NEXT:PRINT
270 :
280 REM LOTTO-3
290 :
300 CLR:DIMA(90):FORI=1TO90:A(I)=0:NEXT
310 PRINT" LOTTOSZAMOK:";:FORI=1TO5
320 J=1+90*RND(0):IFA(J)>0THEN320
330 A(J)=1:NEXT
340 FORI=1TO90:IFA(I)>0THENPRINTI;
350 NEXT:PRINT
```

A 4. és az 5. lista a megoldásban közölt folyamatábrák kódolásai.

```
0 REM 4. LISTA
70 :
80 REM RENDEZETT ADATBEVITEL
90 :
100 CLR:DIMA$(200):INPUT"■■ MI AZ ELSO NEV";A$(1):N=1:DO
110 INPUT" KI A KOVETKEZO";A$:IFLEFT$(A$,1)="#"THENEXIT
120 I=0:J=N+1
130 K=INT((I+J)/2):IFA$<A$(K)THENJ=K:ELSEIFA$>A$(K)THENI=K:ELSEPRINT"VAN MAR!":G
OTO110
140 IFJ-I>1THEN130
150 FORI=NTOJSTEP-1:A$(I+1)=A$(I):NEXT:N=N+1:A$(J)=A$:IFN=200THENEXIT:ELSELOOP
160 :
170 :
180 REM KIRAS
190 :
200 PRINT"■":FORI=1TON:PRINTI"■■. "A$(I):NEXT


0 REM 5. LISTA
10 :
20 REM LINEARIS KERESES
30 :
40 REM BEVITEL
50 :
60 INPUT"■■ HANY ADAT LESZ";N:IFN<2THEN60
70 FORI=1TON:PRINTI"■■. NEV";:INPUTA$(I):NEXT:PRINT"■ KERESES■"
80 INPUT"  MIT KERESSEK";A$
90 :
100 FORI=1TON:IFA$>A$(I)THENNEXT:I=1:GOTO120
110 IFA$<A$(I)THENI=1:ELSEJ=I:FORI=J+1TON:IFA$<A$(I)THEN:ELSENEXT
120 FORI=I-2TOI:NEXT
130 IFITHEN:FORJ=JTOI:PRINTJ"■■. "A$(J):NEXT:PRINT:ELSEPRINT"■ NINCS!"
```

A logaritmikus keresésre az eddigiektől némileg eltérő megoldás látható a 6. listán. Érdemes az előzőkkel összevetni!

```
0 REM 6. LISTA
10 :
20 REM LOGARITMUSOS KERESES
30 :
40 REM BEVITEL
50 CLR:INPUT"■■ HANY ADAT LESZ";N:IFN<2THEN50
60 DIMA$(N):FORI=1TON:PRINTI"■■. NEV";:INPUTA$(I):NEXT
70 :
80 PRINT"■ KERESES■":INPUT"A KERESETT";A$
90 :
100 I=1:J=N:REM ÜRES HALMAZ ELLENORZES:
110 IFI>JTHENPRINT" NINCS AZ ADATOK KOZOTT!":END
120 K=INT((I+J)/2)
130 IFA$=A$(K)THENPRINTA$(K);" INDEXE: ";K:END
140 IFA$<A$(K)THENJ=K-1:GOTO110
150 I=K+1:GOTO110
```

A 7. lista egy többé-kevésbé komplett, névsor készítésére szolgáló program. Lemezegységet vagy kazettás egységet kezelő rutinnal kiegészítve könnyen „összkomfortos" programmá alakítható.

```
0 REM 7. LISTA
10 :
20 REM NEVSOR
30 :
40 DIMA$(200):GOSUB800 CIMLAPOK:B$="ADATBEVITEL.":GOSUB700 FELIRAT
50 GOSUB200 BEVITEL:IFN<2THEN110
60 B$="A BEVITELNEK VEGE,":C$="KIIROM AZ ADATOKAT.":GOSUB700 FELIRAT
70 GOSUB400 KIIRAS:PRINT"■■■■■■  KELL JAVITANI (I/N)?":GOSUB110 0I/N
80 IFA$="I"THENGOSUB100 0JAVITAS:C$="AZ ADATOK:":GOSUB700 FELIRAT:GOTO70
90 PRINT"■■■■■■■■■■■■A RENDEZES FOLYIK.":GOSUB300
100 GOSUB500 BILL.:C$="A NEVSOR:":GOSUB700 FELIRAT:GOSUB400 KIIRAS
110 PRINT"■■■■■■■■■■","NYOMTASSAM (I/N)?":GOSUB110 0I/N
120 IFA$="I"THENPRINT"■■■■■■■■■■","NYOMTATOK!":GOSUB900 NYOMTATAS
130 PRINT"■■■■■■■■■■■■KOSZONOM A FIGYELMET!■■■■■■■"
140 END
150 :
200 PRINT"■■ISMERT AZ ADATOK SZAMA (I/N)?":PRINT
210 GOSUB110 0I/N:IFA$="I"THENINPUT" HANY ADAT LESZ";N:GOTO230
220 N=200:PRINT" LEGFLJEBB 200 ADATOT OLVASOK BE, AZ":PRINT" ADATVEGJEL: '#'.":P
RINT
230 FORI=1TON:GOSUB600 SORSZ.:INPUT"ADAT:";A$:IFLEFT$(A$,1)<>"#"THENA$(I)=A$:NEX
T
240 N=I-1:RETURN:BEVITEL VEGE
250 :
300 FORI=1TON-1:A$=A$(I):K=I
310 FORJ=I+1TON:IFA$>A$(J)THENA$=A$(J):K=J
320 NEXT:A$(K)=A$(I):A$(I)=A$:NEXT
330 RETURN:RENDEZES VEGE
340 :
400 PRINT"■":FORI=1TON
410 GOSUB600 SORSZ.:PRINTA$(I):IF20*INT(I/20)=ITHENGOSUB500 BILL.:PRINT"■"
420 NEXT:IF20*INT(N/20)<>NTHENGOSUB500 BILL.
430 RETURN:KIIRAS VEGE
440 :
500 PRINT"■";:FORK=1TO7:PRINT"■■■■";:NEXT:PRINT"■■<NYOMJ LE EGY BILLENTYUT!>";
510 FORK=0TO9:GETA$:NEXT
520 GETA$:IFA$=""THEN520
530 RETURN:BILLENTYU VEGE
540 :
600 A$=STR$(I):PRINTSPC(4-LEN(A$));A$;". ";
610 RETURN:SORSZAM VEGE
620 :
700 PRINT"■■■■■■■■■■■■";B$:PRINT"■■■■■■";C$:PRINT"■■■■■■";D$
710 B$="":C$="":D$="":GOTO500 FELIRAT VEGE-BILL.
720 :
800 B$="NEVSOR SZERKESZTO PROGRAM.":C$="A BEGEPELT NEVEKET ABC"
810 D$="RENDBE SZEDI.":GOSUB700 FELIRAT
820 B$="A PROGRAM AZ ANGOL ABC":C$="BETUIT HASZNALJA, A RENDE-"
830 D$="ZES IS ESZERINT TORTENIK.":GOSUB700 FELIRAT
840 RETURN:CIMLAPOKVEGE
850 :
900 OPEN4,4:CMD4:FORI=1TON
910 GOSUB600 SORSZ.:PRINTA$(I):NEXT:PRINT#4:CLOSE4:RETURN:NYOMTATAS VEGE
920 :
1000 FORI=1TON:PRINT"■■■■":GOSUB600 SORSZAM:PRINTA$(I):PRINT"■  KELL JAVITANI (I
/N)?■"
1010 GOSUB110 0I/N:IFA$="N"THENNEXT:RETURN
1020 GOSUB600 SORSZAM:INPUT"NEV";A$:IFLEFT$(A$,1)="#"THENNEXT:RETURN
1030 A$(I)=A$:NEXT:RETURN:JAVITAS VEGE
1040 :
1100 FORJ=0TO9:GETA$:NEXT
1110 GETA$:IFA$<>"I"ANDA$<>"N"GOTO1110
1120 RETURN:I/N VEGE
```

```
530 A=A*A(0,0):B=B*A(0,1):GOTO600:SZORZAS, ES OSZTAS
540 A=-A(1,0):GOTO560:KIVONAS MAJD OSSZEADAS
550 A=A(1,0)
560 B=A(1,1):A=B*A(0,0)+A*A(0,1):B=B*A(0,1)
590 :
600 U=ABS(B):V=ABS(A)
610 IFU<VTHENV=V-U*INT(V/(U+1)):GOTO610
620 U=U-V*INT(U/V):IFU>0THEN610
630 RETURN:L.N.K.O.
690 :
700 GOSUB800:LS=LEN(STR$(A))-1:LN=LEN(STR$(B))-1:IFA<0THENPRINT"▩ -⌐";
710 L=LS:IFL<LNTHENL=LN:PRINTSPC((L-LS)/2);
720 A$=STR$(A):PRINT" "RIGHT$(A$,LEN(A$)-1)"▩";:FORI=1TO(LS+L)/2:PRINT"▮";:NEXT
730 FORI=1TOL:PRINT"‾";:NEXT:FORI=1TOL:PRINT"▮";:NEXT:PRINT"▩";
740 IFLN<LTHENPRINTSPC((L-LN)/2)
750 A$=STR$(B):PRINTRIGHT$(A$,LEN(A$)-1);SPC((1+L-LN)/2)" ⌐";:RETURN:KIIRAS
790 :
800 A=(2*(A/B<0)+1)*ABS(INT(A)):B=ABS(INT(B)):RETURN:ELOJEL
890 :
900 GOSUB1100
910 PRINT"▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩▩","(NYOMJ LE EGY BILLENTYUT!)";
920 FORI=0TO9:GETA$:NEXT
930 GETA$:IFA$=""THEN930
940 RETURN
990 :
1000 GOSUB1100:FORI=0TO9:GETA$:NEXT
1010 GETA$:IFA$="I"ORA$="N"THENRETURN
1020 GOTO1010
1090 :
1100 GOSUB1200:A$="":B$="":C$="":D$="":RETURN
1190 :
1200 PRINT"⌐▩▩▩";TAB(20-LEN(A$)/2);A$:PRINT"▩";TAB(20-LEN(B$)/2);B$
1210 PRINT"▩";TAB(20-LEN(C$)/2);C$:PRINT"▩";TAB(20-LEN(D$)/2);D$;"▩":RETURN
1290 :
1300 DATA"▩O■SSZEADAS","+"
1310 DATA"▩K■IVONAS","-"
1320 DATA"▩S■ZORZAS","*"
1330 DATA"▩O■SZTAS",":"
```

A 23. lista prímszámokat nyomtat 6000-ig. Ezt azonban csak akkor tudja, ha a felesleges sorokat kitöröljük, mert egyébként nem férnek az adatok a tárba.

```
0 REM 23. LISTA
10 :
20 REM PRIMSZAMOK NYOMTATASA
30 REM 6000-IG HASZNALHATO, DE
40 REM AZ URES SOROKAT TOROLNI KELL!
50 :
60 INPUT"⌐▩ MEDDIG CSINALJAM";N
70 DIMA%(N):FORI=2TON:A%(I)=1:NEXT
80 FORI=2TO1+SQR(N):IFA%(I)>0THENFORJ=2*ITONSTEPI:A%(J)=0:NEXT
90 PRINTI;:NEXT:PRINT:PRINT" KESZ!"
100 J=1:OPEN4,4:CMD4:FORI=2TON
110 IFA%(I)>0THEN:A$=STR$(I):PRINTSPC(6-LEN(A$));A$;:J=J+1:IFJ>12THENPRINT:J=1
120 NEXT:PRINT#4:CLOSE4
```

Így ne!

Végül a 24. lista egy „elrettentő példa”. Ezért (lóerő—kilowatt átszámításra) nem érdemes programot írni, különösen ilyet nem.

Itt kell megjegyezni, hogy az „üres sorok” a programokból elhagyhatók, ezekre vonatkozó hivatkozások nincsenek. Elsősorban a program tagolását szolgálják a tanulás idejére. A szóközök hiányát pedig érdemes megszokni, a már korábban említett előnyökért.

```
0 REM 24. LISTA
10 :
20 REM ************************************
30 REM *   LOERO KILOWATT ATSZAMITO   *
40 REM *        PROGRAMRENDSZER         *
50 REM ************************************
60 REM
70 REM ******** INICIALIZALAS **********
80 GOSUB 550
90 REM **** UGRAS A VEZERLOMODULBA ****
100 GOTO 160
110 REM
120 REM ************************************
130 REM *          VEZERLOMODUL            *
140 REM ************************************
150 REM
160 REM ******* FELIRATOZAS ************
170 GOSUB 650
180 REM ******* ADATBEVITEL ************
190 GOSUB 320
200 REM ******* FELDOLGOZAS ************
210 GOSUB 410
220 REM ******* ADATKIVITEL ************
230 GOSUB 480
240 REM *** A FOLYTATAS LEKEZELESE ****
250 PRINT "■■TOVABB(I,N)      ■■■■■■■";
260 INPUT X$
270 IF X$="N" THEN GOTO 740
280 IF X$="I" THEN GOTO 160
290 PRINT"■■HIBAS VALASZ!!!!!"
300 GOTO 250
310 REM ***** A VEZERLOMODUL VEGE ******
320 REM
330 REM ************************************
340 REM *          INPUT MODUL             *
350 REM ************************************
360 REM
370 PRINT "KEREM A LOERO ERTEKET";
380 INPUT L
390 REM
400 RETURN
410 REM
420 REM ************************************
430 REM *       FELDOLGOZO MODUL           *
440 REM ************************************
450 REM
460 K=L*PAR
470 RETURN
480 REM
490 REM ************************************
500 REM *         OUTPUT MODUL             *
510 REM ************************************
520 REM
530 PRINT "■"L"LOERO ="K"KILOWATT"
540 RETURN
550 REM
560 REM ************************************
570 REM *      INICIALIZALO MODUL          *
580 REM ************************************
590 REM
600 DATA 0.736
610 FOR I=1 TO 1.5 STEP 1
620 READ PAR
630 NEXT I
640 RETURN
650 REM
660 REM ************************************
670 REM *        FELIRATOZO MODUL          *
680 REM ************************************
690 REM
700 PRINT "□"
```

```
710 PRINT "■■■■■■■■■■■■■■■■■■■■LOERO KILOWATT ATSZAMITASOO"
720 REM
730 RETURN
740 STOP
750 END
```

A programokban lévő egyéb megjegyzések úgy helyezkednek el, hogy nem (RETURN után), vagy csak kevéssé (GOSUB és GOTO után) lassítják a programot. Ezt is érdemes végiggondolni.

A programok mindegyike futott, így remélhető, hogy csak kevés nehézséget okoznak, bosszúságot pedig talán egyáltalán nem.

EPILÓGUS

Nem panasz, de nehéz egy könyvet befejezni!

Annyi dolog van, ami kimaradt; ügyes fogások, sok munkát helyettesítő apróságok, tanulságos mintaprogramok, felsorolni is nehéz lenne.

De talán ötletet adni ennyivel is lehet, és minden bizonnyal hasznos a kulcsszavak leírása — még hosszabb távon — is.

Persze hiba is biztosan akad szép számmal, sok ellentmondás lesz a különböző forrásból származó leírások között. Ez biztosan zavaró, de az Olvasónak kitűnő lehetőséget kínál a döntéshez — a gép. Vallassa a gépét, és csak annak higgyen! Ne feledje: a gép értelmezőjét profik írták!

Ha valami nem sikerül elsőre, próbálkozzon újra, felhasználva az egyéb irodalmakat is, hiszen ez az igazi tanulás alapja. Elkeseredni sohasem szabad; vigasztalja az Olvasót is LORD BALFOUR állítása:

„Semmi sem számít túlságosan, és nagyon kevés dolog számít egyáltalában."

A) ADATBEVITELI HIBÁK

Az INPUT-utasítás végrehajtásakor a gép a tétellistának megfelelő számú és típusú értéket vár.

Ha a számváltozóba való beolvasáskor olyan karaktersorozatot adunk válaszként, amely a szóközöktől, az előjeltől és a jegyek számától eltekintve megfelel a gép nyomtatási szintaktikájának, akkor az konvertálódik, esetleg túlcsordulási hiba következik be. Ha a formátum nem ilyen, akkor „? REDO FROM START" üzenet után ismételt adatbevitelre vár a gép. Az ilyen hibaüzenet kivédhető, ha a számot szövegváltozóba visszük, majd a VAL függvénnyel alakíttatjuk számmá.

Ha a szövegváltozóba vesszőt tartalmazó karaktersorozatot akarunk tölteni, akkor azt idézőjelek közé kell tenni, ellenkező esetben két adatként értelmeződik és az „?EXTRA IGNORED" üzenet kiírása után folytatódik a program.

B) HIBAÜZENETEK

1. TOO MANY FILES	Túl sok file-t akartunk megnyitni. Legfeljebb 10 file lehet egyidejűleg nyitva.
2. FILE OPEN	Már megnyitott file-t próbáltunk újra megnyitni.
3. FILE NOT OPEN	A file-t nem nyitottuk meg az I/O művelet kezdeményezése előtt.
4. FILE NOT FOUND	Nem találja a file-t, mert nincs a lemezen, vagy már elérte a kazetta „szalagvég" (EOT) jelét.
5. DEVICE NOT PRESENT	Az eszköz nem áll rendelkezésre.
6. NOT INPUT FILE	Olvasni akartunk írásra megnyitott fileból.
7. NOT OUTPUT FILE	Írni akartunk olvasásra megnyitott fileba.
8. MISSING FILE NAME	Nem adtunk meg filenevet.
9. ILLEGAL DEVICE NUMBER	Hibás eszköznevet adtunk meg: a művelet az adott eszközzel nem hajtható végre.
10. NEXT WITHOUT FOR	Olyan NEXT-utasítást talált, amelyhez (már) nem tartozik FOR (pl.: a már korábban lefutott, vagy a változónéven ilyen ciklusbejegyzés aktuálisan nincs).

11. SYNTAX	Szintaktikus (nyelvtani) hiba.
12. RETURN WITHOUT GOSUB	RETURN-utasítást akartunk végrehajtani érvényes GOSUB-bejegyzés nélkül.
13. OUT OF DATA	Nincs több adat, a READ-utasítást nem tudta végrehajtani.
14. ILLEGAL QUANTITY	A függvényhivatkozást vagy az utasítást az értelmezési tartományba nem konvertálható értékkel akartuk végrehajtani.
15. OVERFLOW	A számolás eredménye nagyobb, mint a legnagyobb ábrázolható szám.
16. OUT OF MEMORY	Megtelt a programtár vagy a verem.
17. UNDEF'D STATEMENT	Nem létező sorra hivatkoztunk.
18. BAD SUBSCRIPT	Az explicit vagy implicit tömbméret-megadást túlléptük.
19. REDIM'D ARRAY	Egy már létező (explicit vagy implicit módon dimenzionált) tömböt akartunk ismét dimenzionálni.
20. DIVISION BY ZERO	Nullával akartunk osztani.
21. ILLEGAL DIRECT	Parancsüzemmódban nem használható utasítás (**GET**, **INPUT**).
22. TYPE MISMATCH	Típuskeveredés. Sztring helyett szám szerepel, vagy fordítva, és nem konvertálható.
23. STRING TOO LONG	Túl hosszú szöveg. Egy sztring hossza legfeljebb 255 lehet, a billentyűzetről bevitt programsor, ill. adat legfeljebb 8Ø karakterből állhat.
24. FILE DATA	Típuskeveredés kazettáról vagy lemezről származó (beolvasott) adattal.
25. FORMULA TOO COMPLEX	Túl bonyolult kifejezés.
26. CAN'T CONTINUE	A végrehajtás nem folytatható, mert parancsüzemmódban állt meg, vagy javítottuk a programot (törlődtek a változók) vagy még nem futtattuk.
27. UNDEF'D FUNCTION	Nem értelmezett (nem létező felhasználói) függvényre akartunk hivatkozni.
28. VERIFY	File-ellenőrzéskor talált file-hiba.
29. LOAD	Hibás betöltés.
30. BREAK	A programot leállítottuk a RUN/STOP-billentyűvel.
31. CAN'T RESUME	Hibarutint akartunk lezárni anélkül, hogy hibaállapot következett volna be.
32. LOOP NOT FOUND	DO-ciklusból való kiugráskor nem talált LOOP-utasítást.
33. LOOP WITHOUT DO	LOOP-utasítást akartunk végrehajtani érvényes DO-bejegyzés nélkül.
34. DIRECT MODE ONLY	Csak parancsüzemmódban használható.

35. NO GRAPHICS AREA	Nincs grafikus terület, grafikus utasítást akartunk végrehajtani grafikus képernyő definiálása nélkül.
36. BAD DISC	Lemezhiba (hardver).

C) FOGLALT VÁLTOZÓK

DS: Lemezes rendszer állapotváltozója.
DS$: Lemezes rendszer szöveges állapotváltozója.
EL: Az utolsó hibát okozó sor sorszáma. Hiba nélküli értéke 65 535.
ER: Az utolsó hiba kódja. Hiba nélküli értéke −1.
ST: I/O műveletek állapotváltozója.
TI: A rendszer belső órájának változója.
TI$: A belső óra által befolyásolt szövegváltozó, amely két-két jegyen ábrázolja az aktuális óra-, perc- és másodpercértéket.

D) A KULCSSZAVAK BELSŐ KÓD SZERINTI SORRENDJE

128	80	END	156	9C	CLR
129	81	FOR	157	9D	CMD
130	82	NEXT	158	9E	SYS
131	83	DATA	159	9F	OPEN
132	84	INPUT #	160	A0	CLOSE
133	85	INPUT	161	A1	GET
134	86	DIM	162	A2	NEW
135	87	READ	163	A3	TAB(
136	88	LET	164	A4	TO
137	89	GOTO	165	A5	FN
138	8A	RUN	166	A6	SPC(
139	8B	IF	167	A7	THEN
140	8C	RESTORE	168	A8	NOT
141	8D	GOSUB	169	A9	STEP
142	8E	RETURN	170	AA	+
143	8F	REM	171	AB	−
144	90	STOP	172	AC	*
145	91	ON	173	AD	/
146	92	WAIT	174	AE	↑
147	93	LOAD	175	AF	AND
148	94	SAVE	176	B0	OR
149	95	VERIFY	177	B1	>
150	96	DEF	178	B2	=
151	97	POKE	179	B3	<
152	98	PRINT #	180	B4	SGN
153	99	PRINT	181	B5	INT
154	9A	CONT	182	B6	ABS
155	9B	LIST	183	B7	USR

184	B8	FRE
185	B9	POS
186	BA	SQR
187	BB	RND
188	BC	LOG
189	BD	EXP
190	BE	COS
191	BF	SIN
192	C0	TAN
193	C1	ATN
194	C2	PEEK
195	C3	LEN
196	C4	STR$
197	C5	VAL
198	C6	ASC
199	C7	CHR$
200	C8	LEFT$
201	C9	RIGHT$
202	CA	MID$
203	CB	GO
204	CC	RGR
205	CD	RCLR
206	CE	RLUM
207	CF	JOY
208	D0	RDOT
209	D1	DEC
210	D2	HEX$
211	D3	ERR$
212	D4	INSTR
213	D5	ELSE
214	D6	RESUME
215	D7	TRAP
216	D8	TRON
217	D9	TROFF
218	DA	SOUND
219	DB	VOL
220	DC	AUTO
221	DD	PUDEF
222	DE	GRAPHIC
223	DF	PAINT
224	E0	CHAR
225	E1	BOX
226	E2	CIRCLE
227	E3	GSHAPE
228	E4	SSHAPE
229	E5	DRAW
230	E6	LOCATE
231	E7	COLOR
232	E8	SCNCLR
233	E9	SCALE
234	EA	HELP
235	EB	DO
236	EC	LOOP
237	ED	EXIT
238	EE	DIRECTORY
239	EF	DSAVE
240	F0	DLOAD
241	F1	HEADER
242	F2	SCRATCH
243	F3	COLLECT
244	F4	COPY
245	F5	RENAME
246	F6	BACKUP
247	F7	DELETE
248	F8	RENUMBER
249	F9	KEY
250	FA	MONITOR
251	FB	USING
252	FC	UNTIL
253	FD	WHILE
255	FF	π

E) A KULCSSZAVAK FUNKCIÓ SZERINTI CSOPORTOSÍTÁSA

PARANCSOK

AUTO
CONT
DELETE
DIRECTORY
DLOAD
DSAVE
GOTO
HELP
KEY
LIST
LOAD
MONITOR
NEW
RENUMBER
RUN
SAVE
VERIFY

HIBAKERESÉS, BELÖVÉS; HIBAKEZELÉS

CONT
END
ERR$
GOSUB
GOTO
HELP
KEY
LIST
MONITOR
NEXT
PEEK
POKE
RENUMBER
RESUME
RUN
STOP
TRAP
TROFF
TRON

PROGRAMSZERKEZET, VEZÉRLÉSÁTADÁS

DO
ELSE
END
EXIT
FOR
GO
GOSUB
GOTO
IF
LOOP
NEXT
ON
RESUME
RETURN
STOP
THEN
TRAP
UNTIL
WHILE

NYOMTATÁS (KÉPERNYŐRE, NYOMTATÓRA)

CHAR
CMD
INPUT
KEY
LIST
OPEN
POS
PRINT
PRINT #
PUDEF
SCNCLR
SPC(
TAB(
USING

Grafika

BOX
CHAR
CIRCLE
CLR
COLOR
DRAW
GRAPHIC
GSHAPE
LOCATE
PAINT
RCLR
RDOT
RGR
RLUM
SCALE
SCNCLR
SSHAPE

NUMERIKUS FÜGGVÉNYEK

ABS
ATN
COS
EXP
FRE
INT
JOY
LOG
PEEK
POS
RCLR
RDOT
RGR
RLUM
RND
SGN
SIN
SQR
TAN
USR

Különleges függvények

ASC
CHR$
DEC
ERR$
FRE
HEX$
INSTR
JOY
LEFT$
LEN
MID$
PEEK
POS
RCLR
RDOT
RGR
RIGHT$
RLUM
RND
SPC(
STR$
TAB(
USR
VAL

SZÖVEGKEZELÉS

ASC
CHR$
DEC
ERR$
HEX$
INSTR
LEFT$
LEN
MID$
RIGHT$
STR$
VAL
+

MŰVELETEK

AND
NOT
OR
<
>
=
+
−
*
/
↑

PERIFÉRIÁK, ÁLLOMÁNYOK

BACKUP
CLOSE
CMD
COLLECT
COPY
DIRECTORY
DLOAD
DSAVE
GET
HEADER
INPUT #
LOAD
OPEN
PRINT
PRINT #
RENAME
SAVE
SCRATCH
VERIFY

Utasításban használt kulcsszavak

CLR
ELSE
FN
GOSUB
GOTO
KEY
ON
STEP
THEN
TO
UNTIL
USING
WHILE

IRODALOMJEGYZÉK

Bencsikné Takács Márta: Feladatgyűjtemény C16-os számítógéphez általános iskolásoknak. Budapest, NOVOTRADE, 1986.

COMMODORE 16 Felhasználói kézikönyv. Fordította és szerkesztette: *Tory Kálmán.* Budapest, NOVOTRADE.

Kőhegyi János (szerk.): Ismerd meg a BASIC nyelvjárásait! COMMODORE 64 COMMODORE VIC 20 SHARP PC-1500. Budapest, Műszaki Könyvkiadó, 1986.

Lőcs Gyula: A BASIC és a Kíváncsi. Budapest, Tankönyvkiadó, 1985.

Lőcs Gyula: A BASIC és a Kíváncsi Feladatgyűjtemény. Budapest, Tankönyvkiadó, 1986.

PLUSZ/4-es Felhasználói kézikönyv. Budapest, NOVOTRADE, 1986.

Dr. Úry László: COMMODORE C16 BASIC és Felhasználói kézikönyv. Budapest, LSI Alkalmazástechnikai Tanácsadó Szoltálat, 1985.

Weltner, H. T.: További Tippek és Trükkök a Commodore 64-eshez. Budapest, DATA BECKER—NOVOTRADE. 1986.

49 Ft

BUDAPEST
MŰSZAKI KÖNYVKIADÓ